Safe Exploration Algorithms for Reinforcement Learning Controllers

Tommaso Mannucci, *Graduate Student Member, IEEE*, Erik-Jan van Kampen, Cornelis de Visser, and Qiping Chu

***Abstract*— Self-learning approaches, such as reinforcement learning, offer new possibilities for autonomous control of uncertain or time-varying systems. However, exploring an unknown environment under limited prediction capabilities is a challenge for a learning agent. If the environment is dangerous, free exploration can result in physical damage or in an otherwise unacceptable behavior. With respect to existing methods, the main contribution of this paper is the definition of a new approach that does not require global safety functions, nor specific formulations of the dynamics or of the environment, but relies on interval estimation of the dynamics of the agent during the exploration phase, assuming a limited capability of the agent to perceive the presence of incoming fatal states. Two algorithms are presented with this approach. The first is the Safety Handling Exploration with Risk Perception Algorithm (SHERPA), which provides safety by individuating temporary safety functions, called backups. SHERPA is shown in a simulated, simplified quadrotor task, for which dangerous states are avoided. The second algorithm, denominated OptiSHERPA, can safely handle more dynamically complex systems for which SHERPA is not sufficient through the use of safety metrics. An application of OptiSHERPA is simulated on an aircraft altitude control task.**

***Index Terms*— Adaptive controllers, model-free control, reinforcement learning (RL), safe exploration.**

I. Introduction

IN ENGINEERING, classic control schemes such as PID still enjoy widespread use. This can be partially explained by the amount of effort needed to provide affordable yet efficient dynamic models of complex platforms. In the wake of this consideration, special attention in the control community has been dedicated to control schemes, which require less precise knowledge of a model to achieve satisfactory performances. Robust control [1] constitutes an example of controller design developed to tolerate modeling error while guaranteeing a lower bound on performance. Adaptive control represents a promising field in developing new controllers with increased performance and reduced model dependence [2], [3]. A model-free option amidst adaptive control is reinforcement learning (RL).

RL is a knowledge-based control scheme that mimics animal development [4]. At any given moment, an animal receives an array of internal and external stimuli that form its *situation*, with the *behavior* dictating the reaction to each of them. Correct reactions generate a positive chemical discharge that reinforces the behavior, whereas unsuccessful ones lead to anguish that disproves it. This has an equivalent in RL: stimuli constitute the *plant* or *system*; the animal is the *agent* following a temporary behavior, i.e., *a policy*. Last, the chemical reaction is represented by a numerical feedback called the *reward*.

From a theoretical point of view, RL has evolved in time to guarantee minimal level of performance. Selected algorithms were proven to be probably approximately correct [5], and proofs of near-optimality and optimality for both discrete and continuous [6] applications were found. RL has also proved its worth in combination with neural networks (NNs) in the development of neurodynamic programming [7], [8], where the approximation power of NNs is utilized to efficiently represent both the value and the policy, thus reducing the "curse of dimensionality" [9]. In recent years, multiple successful applications [10]–[13] utilize RL as a framework for learning, in particular *model-free* RL [14]–[16].

In model-free RL, there is no need of a model of the plant: learning starts with an *exploratory* phase, during which, rather than following the best policy based on current knowledge, new actions are tried in order to find the most rewarding actions and iteratively correcting its policy. The agent transitions gradually from exploring to *exploiting*, i.e., performing actions suggested by its best policy. The transition must be handled with care. Transitioning before the best policy is converged results in suboptimal behavior due to lack of knowledge of the environment, but transitioning long after convergence unnecessarily delays the application of the best policy, thus affecting performance. This conflict between gathering and use of knowledge is called *exploration–exploitation dilemma*.

Whenever an agent attempts an inappropriate action, the consequent penalty acts as a negative reinforcement, and the wrong behavior is progressively discouraged, until it is no longer adopted. This permits to accommodate unwanted, unsafe actions within the canonical RL framework. However, consider a hostile environment where the consequences of wrong actions are not limited to "bad" performance, but include long term effects that cannot be compensated by more profitable exploitation later on. As one wrong action can result in unrecoverable effects, such an environment poses a *safety-exploration* dilemma, especially for a model-free approach. The goal of this paper is to avoid such occurrences during learning, thus achieving *safe exploration* [17].

In the literature, a widely adopted approach to safety consists of assigning negative reward for undesired transitions, such that the most reliable policy maximizes the minimal sum of reward in the presence of uncertainties and stochasticity. Safety is therefore embedded into policy performance. This

Manuscript received March 18, 2016; revised October 28, 2016; accepted January 6, 2017. Date of publication February 6, 2017; date of current version March 15, 2018.

The authors are with the Control and Simulation Division, Faculty of Aerospace Engineering, Delft University of Technology, 2629HS Delft, The Netherlands.(e-mail: t.mannucci@tudelft.nl; e.vankampen@tudelft.nl; c.c.devisser@tudelft.nl; q.p.chu@tudelft.nl).

Digital Object Identifier 10.1109/TNNLS.2017.2654539

worst case or *minimax* [18] approach belongs to the *optimization criterion* of safety [19]. Under this criterion, assuming a sufficiently large penalty for unsafe transitions, the optimal policy is also the safest. Methods for policy improvement with this approach have also been designed [20]. Alternatively, by assigning negative reward, the variance of the return can be taken into account by adopting *risk-sensitivity* approaches [21]–[24]. Several techniques exist to implement both minimax and risk-sensitive methods [25]; however, there are limitations as far as exploration is considered. Including safety as part of the reward can generate a conflict between purely performance-based reward and safety-based reward if the penalty for unsafe transitions is not correctly assigned. Also, the optimization criterion can be effective in preventing harmful situations, but requires previous knowledge of the probability of risk for the state-action space, which is in general the result of exploration itself. A different solution is to include safety in the *exploration process* itself. García Fernández [19] differentiate three different approaches: "providing initial knowledge," directing the learning in its initial stage toward more profitable and safer regions of the state space [26]; "deriving a policy from demonstrations" by means of *learning from demonstration* [27]; "providing teacher advice" by including an external teacher that can interrupt exploration and provide expert knowledge, or that the agent can consult when confronted with unexpected situations [28]. An alternative implementation of this solution is *risk-directed* exploration. With this approach, the agent's choice of actions is aided by an appropriate *risk metric* [29] acting as an exploration bonus toward the safer regions of the environment.

Among algorithms that directly avoid unsafe transitions, [30] relies on an *a priori* known safety function (acting as a go/no-go decision maker over feasible actions) and a fixed backup policy valid in all workspace. A similar approach is taken in [31], with the difference that the safety function is obtained through a "cautious simulator." The simulator must correctly label unsafe states, but is allowed to mislabel safe states as unsafe: it is assumed that an experienced human operator can force the system into a mislabeled safe state. In [32], variable amount of perturbation is introduced in a given safe but inefficient baseline controller, so that discovery of new trajectories for task completion is possible, taking a certain amount of risk. These techniques share the need of a guaranteed safe controller, simulators, or backup policy in order to prevent catastrophic exploration when facing critical decisions. Moldovan and Abbeel [33] define safety in terms of ergodicity of the exploration, and introduce an algorithm that relies on believes of the system, but not on a predefined baseline policy or safe controller.

This paper adopts a different strategy for safe exploration that does not rely on *a priori* known safety functions, cautious simulators, or in an explicit ergodicity of the system. Instead, a temporary safety function is generated at each time in the form of *backups*. A backup is essentially an escape route: a control sequence that can bring the system in a close neighborhood of a state that the agent already visited in the past. By resorting to backups, this strategy eliminates the need for the above prerequisites. This is particularly apt for RL tasks where the agent's knowledge is very limited, which makes these prerequisites more demanding.

Two algorithms implementing the strategy are presented in this paper, and compared to similar preexisting ones. The first algorithm is the Safety Handling Exploration with Risk Perception Algorithm (SHERPA). Given a user-defined exploratory policy and an uncertain model of the system, SHERPA searches backups satisfying a *closeness condition*. If the search fails, SHERPA replaces the policy action with a safer alternative, effectively acting as a "filter" with respect to the exploratory policy. SHERPA is effective in all those cases where backups can be found with reasonable ease. For more challenging tasks, a second version called OptiSHERPA is proposed, which uses *metrics* to assess the safety of all available actions. Additionally, OptiSHERPA includes a dedicated evasion strategy that relies on the current belief on the state space to minimize risks while at the same time avoiding impending dangers. While the "filtering" effect of both algorithms could be applied in diverse contexts, it is in RL that they prove more useful, due to the usual lack of information about the task being performed.

For the most general scenario, coverage of the state space, and convergence to the optimal policy cannot be guaranteed when using the proposed algorithms, which promote safety of exploration more than its efficiency (e.g., does not follow the "optimism in the face of uncertainty" [34] criteria). However, the formulation of the proposed algorithms is flexible enough to address these drawbacks without major modifications, e.g., by relaxing the constraints for the individuation of a backup.

Two applications of the algorithms are shown in the form of simulations. SHERPA is applied to a simulated quadrotor UAV exploring an indoor environment. OptiSHERPA is used to control the elevator deflection of a fighter aircraft exploring its flight envelope. Both agents have a stochastic exploratory policy: the goal of the algorithms is then to enforce safety during exploration.

II. Fundamentals

This section will expand the problem of safe exploration. First, a brief overview of the fundamentals of RL will be provided. The motivation will be presented and the problem statement will be formalized. Finally, the assumptions of this method will be summarized.

A. Problem Statement

Classic RL is defined on the Markov decision process (MDP) framework. An MDP schematically represents a task for an agent in an environment, and consists of a tuple of five elements: state, action, transition, reward, and discount. Set $\mathcal{S}$ contains the states $\boldsymbol{x}$ of the environment. Set $\mathcal{A}$ is the set of *actions* $\boldsymbol{u}$ that the agent can select. $\mathcal{D}$ is a mapping $\mathcal{S} \times \mathcal{A} \times \mathcal{S} \to [0, 1]$ that $\forall \boldsymbol{x}, \boldsymbol{x}' \in \mathcal{S}$ and $\forall \boldsymbol{u} \in \mathcal{A}$ assigns to triplet $(\boldsymbol{x}, \boldsymbol{u}, \boldsymbol{x}')$ the probability of the environment transitioning from state $\boldsymbol{x}$ to state $\boldsymbol{x}'$ given action $\boldsymbol{u}$. Function $\mathcal{R} : \mathcal{S} \times \mathcal{S} \to \mathbb{R}$ maps all transitions to a scalar r, the *reward*, which indicates the immediate benefit of the transition with respect to accomplishing the task. A policy $\pi : \mathcal{S} \times \mathcal{A} \to [0, 1]$ is a mapping that $\forall \boldsymbol{x}$ assigns a probability of selecting action $\boldsymbol{u}$, describing the behavior of the agent. The goal of the agent is to learn an optimal policy π^* that maximizes the sum of expected reward $J^\pi(\boldsymbol{x})$ subjected to a discount γ which privileges short term reward. J is the *value* of $\boldsymbol{x}$ with π.

Various algorithms [35] exist to solve this problem, all of which rely on the concept of *exploration*. Exploration consists in performing different actions in different states, observing the consequent reward, and using this knowledge to progressively define a policy π which maximizes the total expected reward. Under certain conditions, exploration can eventually yield policy π^* or its approximation [6].

A *fatal occurrence* is defined as an unacceptable condition for the agent; for example if the agent is harmed, e.g., a crash or a failure, or if it cannot proceed further in its task. Safe exploration consists in preventing fatal occurrences.

B. Definitions and Assumptions

In order to define the approach to safe exploration, this section will introduce multiple assumptions: a framework on how to represent fatal occurrences in physical agents, a limited prediction capability, denominated *risk perception*, and a model for the computation of uncertain dynamics. The dynamics represented by $\mathcal{D}$ define the set $\mathcal{T}$ of feasible transitions $\tau = (\boldsymbol{x}, \boldsymbol{u}, \boldsymbol{x}')$, $\boldsymbol{x}, \boldsymbol{x}' \in \mathcal{S}$, $\boldsymbol{u} \in \mathcal{A}$. Let $\mathcal{T}_{\text{fat}}$ represent the subset of those transitions generating fatal occurrences. The following will be assumed.

Assumption 1: If $\exists \bar{\tau} = (\bar{\boldsymbol{x}}, \bar{\boldsymbol{u}}, \boldsymbol{x}') \in \mathcal{T}_{\text{fat}}$, and $\exists \boldsymbol{x} \in \mathcal{S}$, $\exists \boldsymbol{u} \in \mathcal{A}$, $\exists \tau' = (\boldsymbol{x}, \boldsymbol{u}, \boldsymbol{x}') \in \mathcal{T}$, then $\tau' \in \mathcal{T}_{\text{fat}}$.

This assumption is not new to the literature [36] and can be considered an extension of the Markov property to fatal occurrences: these are inherently related to *fatal states* and not to actions. Any action leading to the same state or condition would result in the same fatal occurrence. This allows to define the *fatal state space* (FSS) as

Definition 1 (Fatal State Space):

$$\text{FSS} = \{\boldsymbol{x}' | \forall \boldsymbol{x} \in \mathcal{S},\ \forall \boldsymbol{u} \in \mathcal{A},\ \tau = (\boldsymbol{x}, \boldsymbol{u}', \boldsymbol{x}') \in \mathcal{T}_{\text{fat}}\}$$

and analogously the *safe state space* (SSS) as

Definition 2 (Safe State Space):

$$\text{SSS} = \mathcal{S} \setminus \text{FSS}.$$

Complex systems may present multiple modalities of fatal occurrences; these will be informally defined as *risks*. It is reasonable to assume that only a subset of state components will be involved for each risk. Given the state space $\mathcal{S} \subseteq \mathbb{R}^n$, define as the *restricted state space* (RSS) of a risk the space defined by those components involved in the risk. Consider for example an aircraft for which two risks are defined: hitting the ground and getting damaged by the aerodynamic forces. The ground risk can be related to the position of the agent with respect to the ground, whereas the aerodynamic risk can be related to wind speed, deflections, and attitudes. Different features of the state vector are involved for each risk. The first RSS is the space of all possible positions. The second RSS is the space of all possible combinations of wind speed, deflections, and attitudes. It will be assumed that the possible modes of fatal occurrence are known.

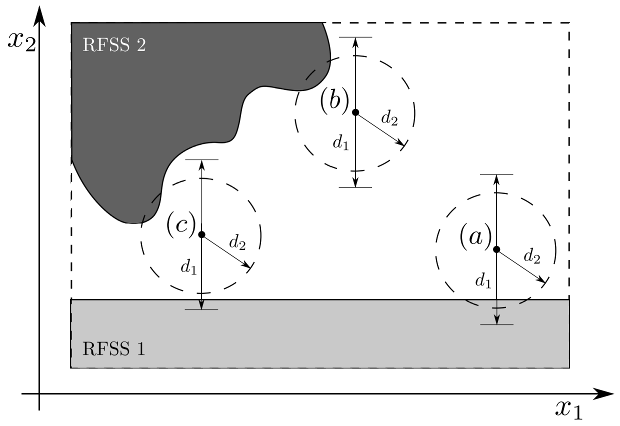


Fig. 1. Agent exploring a state space (bold dotted rectangle) with two different RFSS. The first cause of fatality is independent of the value of component x_1, and an Euclidean distance d_1 is provided for the risk perception. A second cause of fatality is dependent on both components of the state so that an Euclidean distance d_2 in the whole state space is given. (a) RFSS1 is in reach. (b) RFSS2 is in reach. (c) Both RFSSs are in reach. In all three cases, the agent receives an identical warning.

Assumption 2: For each risk, the agent has *a priori* knowledge of the relative RSS.

Such a formulation arguably comes natural to a designer, since it expresses the risks involved in the exploration, and is also very high level, so that the designer is not required to consider dangerous conditions but only those that are certainly fatal. Those elements of an RSS that are also fatal form the *a priori* unknown *restricted fatal state space* (RFSS) (see Fig. 1). Now that a structure for the fatal occurrences is defined, assumption 3 will introduce a mean of prevention of risks, providing closure to the problem definition.

Assumption 3 (Risk Perception): Let RSS_k be the restricted state space of the kth risk, $k \in \{1, \ldots, m\}$. Let $\boldsymbol{x}_k$ be the projection of state $\boldsymbol{x} \in \mathcal{S}$ to RSS_k, and $\boldsymbol{f}_k \in \text{RFSS}_k$ the nearest fatal state to $\boldsymbol{x}_k$. Then, $\forall k \in \{1, \ldots, m\}$, $\exists \epsilon_k \in \mathbb{R}$ such that at each time step, the value of the boolean function

$$\|\boldsymbol{x}_1 - \boldsymbol{f}_1\| \leq \epsilon_1 \vee \cdots \vee \|\boldsymbol{x}_m - \boldsymbol{f}_m\| \leq \epsilon_m \tag{1}$$

is known, where $\| \ \|$ is the canonical 2-norm.

The risk perception can be represented by a unknown mapping $W : \mathcal{S} \to \{0, 1\}$, $W(\boldsymbol{x})$ equal to 1 if (1) is true and 0 otherwise. Risk perception is a strong assumption; however, the following should be considered. When performing exploration in a dangerous environment, the agent cannot predict if the next state will be fatal or not without any form of knowledge of the FSS. Risk perception constitutes a plausible source of knowledge, arguably more valid than *a priori* assumptions on fatal states distributions. Being a form of local, sensor-based knowledge, risk perception allows to react to the insurgence of risks while retaining a sufficient level of abstraction and generality.

Consider the continuous-time version of an MDP. Indicating by $\boldsymbol{\sigma}(\boldsymbol{x}(t_0), \boldsymbol{u}(t), t)$ the state of the system at time $t > t_0$ when applying generic action history $\boldsymbol{u}(\boldsymbol{x}(t_0), t)$

$$\boldsymbol{\sigma}(\boldsymbol{x}(t_0), \boldsymbol{u}(t), t) = \boldsymbol{x}(t_0) + \int_{t_0}^{t} \dot{\boldsymbol{x}}(\boldsymbol{x}, \boldsymbol{u}(\tau)) d\tau \tag{2}$$

the set L of the lead-to-fatal (LTF) states is

Definition 3 (Lead-to-Fatal States):

$$L = \{\boldsymbol{x} | \forall \boldsymbol{u}(x(t_0), t), \exists t : \boldsymbol{\sigma}(x(t_0), \boldsymbol{u}(t), t) \in \text{FSS}\}. \tag{3}$$

LTF states are those that do not belong to the FSS, but evolve in the FSS with probability one. They can be seen as an equivalent in the generalized states space of the *inevitable collision states* introduced in [37], and as the *supercritical* states defined in [30] for a deterministic MDP.

In model-based RL, the controller has an adaptive model of the environment that allows off-line training and online system identification. Instead, in model-free RL (e.g., Q-learning [38]) there is no off-line training—the policy refinement is based on online training and no model is generated. An intermediate position between these two instances will be adopted in this paper by introducing the following definition.

Definition 4 (Bounding Model): Given dynamics $\mathcal{D}$ yielding transition set $\mathcal{T}$, model $\boldsymbol{\Delta}(\boldsymbol{x}, \boldsymbol{u})$ is bounding for $\mathcal{D}$ if and only if

$$\forall \tau = \{\boldsymbol{x}, \boldsymbol{u}, \boldsymbol{x}'\} \in \mathcal{T}; \quad \boldsymbol{x}' \in \boldsymbol{\Delta}(\boldsymbol{x}, \boldsymbol{u}) \tag{4}$$

i.e., a model is bounding if it predicts *at least* all feasible transitions. Such a model is not used to perform off-line policy improvements but only to predict the boundaries of the dynamical evolution of the agent. It will be assumed that both real and bounding models are continuous, deterministic, and time-invariant.

III. SHERPA

This section presents SHERPA. In order to properly explain the algorithm, *interval analysis* (IA) will be briefly introduced as a mean of obtaining bounding models [39]. The mathematical framework will then be expanded and refined as well. The algorithm will then be discussed in detail. Finally, the section will present an application.

A. Interval Analysis

An interval $[\boldsymbol{x}] = [\underline{\boldsymbol{x}}, \overline{\boldsymbol{x}}]$ is the set $\{\boldsymbol{x} | \underline{x}_i \leq x_i \leq \overline{x}_i, i \leq n\}$, $\underline{\boldsymbol{x}} = (\underline{x}_1, \ldots, \underline{x}_n)^T$, $\overline{\boldsymbol{x}} = (\overline{x}_1, \ldots, \overline{x}_n)^T$, $\underline{x}_i, \overline{x}_i \in \mathbb{R}$. If $\underline{\boldsymbol{x}} = -\overline{\boldsymbol{x}}$, $[\boldsymbol{x}]$ is *symmetric*. In IA, the generic operator $*$ applied to intervals yields all feasible solutions of the same operation applied to elements of the intervals: $\forall \boldsymbol{x} \in [\boldsymbol{x}], \boldsymbol{y} \in [\boldsymbol{y}], \boldsymbol{x} * \boldsymbol{y} \in [\boldsymbol{x}] * [\boldsymbol{y}]$. Consider a parametric formulation of $\mathcal{D}$ as $\mathcal{D}(\boldsymbol{x}, \boldsymbol{u}, \rho_1, \ldots, \rho_k)$, with parameters $\rho_i \in [\rho_i]$ and $[\rho_i]$ known confidence intervals. Following the above definition, $[\mathcal{D}(\boldsymbol{x}, \boldsymbol{u})] = \{\mathcal{D}(\boldsymbol{x}, \boldsymbol{u}, \rho_1, \ldots, \rho_k) | \rho_i \in [\rho_i]\}$ is the bounding model of $\mathcal{D}$. For each $\boldsymbol{x}$ and $\boldsymbol{u}$, $[\mathcal{D}(\boldsymbol{x}, \boldsymbol{u})]$ is also an interval. This makes bounding models relatively simple and very economic in terms of data representation. However, it has the disadvantage of overestimating the *reachable set* of the states, due to an effect known as *dependence problem*, which propagates uncertainty in states and parameters.

B. Background and Algorithm Description

The final goal of the agent is to select at each time t a control $\boldsymbol{u}$ that will keep the system safe, i.e., will avoid the FSS. Safety of a control can be defined as follows.

Definition 5: Control $\boldsymbol{u}(t)$, $t \in [t_1 t_2]$ is **safe** in state $\boldsymbol{x}(t_1)$ if

$$\boldsymbol{\sigma}(\boldsymbol{x}(t_1), \boldsymbol{u}, t) \in \text{SSS} \quad \forall t \in [t_1 t_2].$$

Safe control $\boldsymbol{u}(t)$ is *feasible* if $\forall t$, $\underline{\boldsymbol{u}} \leq \boldsymbol{u}(t) \leq \overline{\boldsymbol{u}}$, where $\underline{\boldsymbol{u}}$ and $\overline{\boldsymbol{u}}$ represent boundaries on control, e.g., saturation. If this condition is strictly satisfied, $\boldsymbol{u}(t)$ is said to be *strictly feasible*. However, multiple limitations such as inertia, control saturation, and holonomic constraints give rise to LTF states, which must also be avoided during exploration, but are not perceivable. For a generic trajectory $\boldsymbol{\sigma}(x(t_0), \mathbf{u}, t)$, the condition of (3) cannot be verified in practice since it would require an infinite time horizon. However, an exception occurs when a trajectory $\boldsymbol{\sigma}$ visits a state $\boldsymbol{x}(\bar{t})$ multiple times. Formally, given a safe and feasible control $\boldsymbol{u}(t)$ with associated trajectory $\boldsymbol{\sigma}(\boldsymbol{x}(t_0), \boldsymbol{u}, t)$, $t \in [t_0 t_1]$ if

$$\exists t', t'' \in [t_0 t_1], t' \neq t'' : \boldsymbol{\sigma}(\boldsymbol{x}(t_0), \mathbf{u}, t') = \boldsymbol{\sigma}_{\boldsymbol{u}}(\boldsymbol{x}(t_0), \mathbf{u}, t'') \tag{5}$$

then there exists a safe and feasible control $\boldsymbol{u}^*(t)$ for $t \in [t_0 \infty)$. It can be seen from (2) that (5) can be rewritten as $\boldsymbol{\sigma}(\boldsymbol{x}(t'), \boldsymbol{u}(t - (t' - t_0)), t') = \boldsymbol{\sigma}(\boldsymbol{x}(t'), \boldsymbol{u}(t - (t' - t_0)), t'')$. Then, $\forall \Delta t \leq (t'' - t')$, $\boldsymbol{\sigma}(\boldsymbol{x}(t''), \boldsymbol{u}(t - (t'' - t') - (t' - t_0)), t'' + \Delta t) = \boldsymbol{\sigma}(\boldsymbol{x}(t'), \boldsymbol{u}(t - (t' - t_0)), t' + \Delta t)$. Since trajectory $\boldsymbol{\sigma}(\boldsymbol{x}(t'), \boldsymbol{u}(t), t' + \Delta t) \in \text{SSS}$, a safe and feasible control exists at least for $t \in [t_0 t'' + (t'' - t')]$. The above can be repeated by replacing t' with t'' and t'' with $t'' + (t'' - t')$, guaranteeing a safe control for $t \in [t_0 t'' + 2(t'' - t')]$. The procedure can be repeated indefinitely, thus demonstrating the above. A particular occurrence of (5) consists in those cases where the system is in *equilibrium*, i.e., $\dot{\boldsymbol{x}} = 0$. Then, define an *ideal backup* from $\boldsymbol{x}_0 = \boldsymbol{x}(t_0)$ to $\boldsymbol{x}(\bar{t})$ as follows:

Definition 6: A feasible, safe control action $\boldsymbol{u}_b(t)$, $t \in [t_0 \bar{t}]$ is an **ideal backup** from $\boldsymbol{x}_0$ to $\boldsymbol{x}(\bar{t})$ if

$$\exists\ t < \bar{t} : \boldsymbol{x}(\bar{t}) = \boldsymbol{x}(t). \tag{6}$$

An ideal backup guarantees safety; however, (6) cannot be verified when taking into account the uncertainties of the bounding model, since it is not possible to exactly predict the next state. The following assumption will be made.

Assumption 4: Let $\boldsymbol{u}(t)$ be a strictly feasible safe control for $\boldsymbol{x}_0 = \boldsymbol{x}(t_0)$ and $t \in [t_0 \bar{t}]$. Then, $\forall \epsilon > 0$, $\exists M \in \mathbb{R}$ and $\boldsymbol{u}^*(t)$ a safe control for $\boldsymbol{x}$: $\|\boldsymbol{x} - \boldsymbol{x}_0\| < \epsilon$ and $t \in [t_0 \bar{t}]$ such that

$$\max_t \|\boldsymbol{u}(t) - \boldsymbol{u}^*(t)\| \leq M, \quad \lim_{\epsilon \to 0} M = 0$$

i.e., in a neighborhood of a state with a strictly feasible safe control, a safe control for any state of the neighborhood can be found by altering the safe control by a finite amount that tends to zero as the neighborhood reduces in size.[1] A weaker definition of a *backup* can then be introduced:

Definition 7: A feasible, safe control action $\boldsymbol{u}_b(t)$, $t \in [t_0 \bar{t}]$ is a **backup** from $\boldsymbol{x}_0$ to $\boldsymbol{x}(\bar{t})$ with reach ϵ if

$$\exists\ t < \bar{t} : \|\boldsymbol{x}(\bar{t}) - \boldsymbol{x}(t)\| \leq \epsilon \tag{7}$$

which in turn allows for the following theorem.

[1]Intuitively, local Lipschitz continuity of $(d\boldsymbol{x}/dt)(\boldsymbol{x}, \boldsymbol{u})$ is required for Assumption 4 to hold. However, the remainder of the paper will consider Assumption 4 as given.

Theorem 1: Let $\boldsymbol{x}_1 = \boldsymbol{x}(t_1) \in \text{SSS}$, let $\boldsymbol{u}_S(t)$ be a strictly feasible safe control for $t \in [t_0 t_1]$, and let $\boldsymbol{u}_I(t)$ be a strictly feasible ideal backup from $\boldsymbol{x}_1$ to $\boldsymbol{x}_2 = \boldsymbol{\sigma}(\boldsymbol{x}_1, \boldsymbol{u}_I, t_2)$. Then, $\exists \boldsymbol{u}_b$ a backup from $\boldsymbol{x}_1$ to $\boldsymbol{x}_2$ with reach $\epsilon > 0$ and a control $\boldsymbol{u}^*(t)$ for $t \geq t_2$ such that

$$\boldsymbol{u}(t) = \begin{cases} \boldsymbol{u}_b(t) & \text{if } t \leq t_2 \\ \boldsymbol{u}^*(t) & \text{otherwise} \end{cases}$$

is a feasible safe control for $t \in [t_1 \infty)$.

Proof: The existence of the ideal backup $\boldsymbol{u}_I(t)$ means that the iterative control action $\boldsymbol{u}_{\text{iter}}(t) = (\boldsymbol{u}_I(t), \boldsymbol{u}_S(t), \boldsymbol{u}_I(t), \ldots)$ is a strictly feasible safe control. The existence of $\boldsymbol{u}_I(t)$ means that $\forall \epsilon$ there exists a backup. Assumption 4 guarantees that there exists a safe control $\boldsymbol{u}(t)$ such that $\forall t \|\boldsymbol{u}(t) - \boldsymbol{u}_{\text{iter}}(t)\| \leq M$ after the application of the backup. Since $\boldsymbol{u}_{\text{iter}}(t)$ is strictly feasible, and since, for $\epsilon \to 0$, $M \to 0$, then $\exists \epsilon$ s.t. $\boldsymbol{u}(t)$ is also feasible. □

Theorem 1 is of limited practical use—since it relies on the existence of $\boldsymbol{u}_I(t)$ and does not specify the actual value of ϵ—but formalizes the observation contained in (5) in the presence of uncertainty. The equilibrium condition can be formalized as follows.

Definition 8: State $\boldsymbol{x}_E \in \text{SSS}$ is an *equilibrium point* for the system if $\exists \boldsymbol{u}_E = \boldsymbol{u}(t_E)$ s.t. $\mathrm{d}\boldsymbol{x}/\mathrm{d}t(\boldsymbol{x}_E, \boldsymbol{u}_E) = \boldsymbol{0}$.

The definition of backup can be revised as follows.

Definition 9: A feasible, safe control action $\boldsymbol{u}_b(t)$, $t \in [t_0 \bar{t}]$ is a **backup** from $\boldsymbol{x}_0$ to $\boldsymbol{x}(\bar{t})$ with reach ϵ if $\exists\, t < \bar{t} : \|\boldsymbol{x}(\bar{t}) - \boldsymbol{x}(t)\| \leq \epsilon$, in which case is said to have *reach* ϵ, or if there exists an equilibrium point $\boldsymbol{x}_E$ s.t. $\|\boldsymbol{x}(t_2) - \boldsymbol{x}_E\| \leq \delta$.

C. Closeness Condition

The goal of SHERPA is to find a feasible backup with a sufficiently small reach, i.e., for which the system reaches a "close" neighborhood of a previously visited state; therefore, a *closeness condition* must heuristically be introduced to implement the method. Once again, IA offers a simple way of interpreting closeness between two points. Before proceeding further, it is convenient to slightly alter the definition of backup to accommodate the use of intervals. Let the bounding model $\boldsymbol{\Delta}$ of $\mathcal{D}$ be its IA extension $[\mathcal{D}]$ with time-discrete formulation, so that $[\boldsymbol{x}_{k+1}] = \boldsymbol{\Delta}([\boldsymbol{x}_k], \boldsymbol{u}_k)$, and let $[\boldsymbol{\epsilon}]$ and $[\boldsymbol{\delta}]$ be two n-dimensional symmetric intervals. Then, a backup can be reformulated as follows.

Definition 10: Let then $\{\boldsymbol{x}_k, \ldots, \boldsymbol{x}_{k+m}\}$ be the trajectory generated by control $\boldsymbol{u}_b = \{\boldsymbol{u}_k, \cdots, \boldsymbol{u}_{k+m}\}$. Then, $\boldsymbol{u}_b$ is a **backup** in interval form for $[\boldsymbol{x}_k]$ if and only if

$$\forall i \in \{1, 2, \ldots, m\}, \quad [\boldsymbol{x}_{k+i}] \subset \text{SSS} \tag{8}$$

and

$$\forall \boldsymbol{x}_{k+m} \in [\boldsymbol{x}_{k+m}], \quad \exists p \leq k : (\boldsymbol{x}_{k+m} - \boldsymbol{x}_p) \in [\boldsymbol{\epsilon}] \tag{9}$$

in which case the backup has reaching interval $[\boldsymbol{\epsilon}]$, or if it exists an equilibrium point $\boldsymbol{x}_E$ such that

$$\forall \boldsymbol{x}_{k+m} \in [\boldsymbol{x}_{k+m}], \quad (\boldsymbol{x}_{k+m} - \boldsymbol{x}_E) \in [\boldsymbol{\delta}] \tag{10}$$

in which case the backup has reaching interval $[\boldsymbol{\delta}]$.

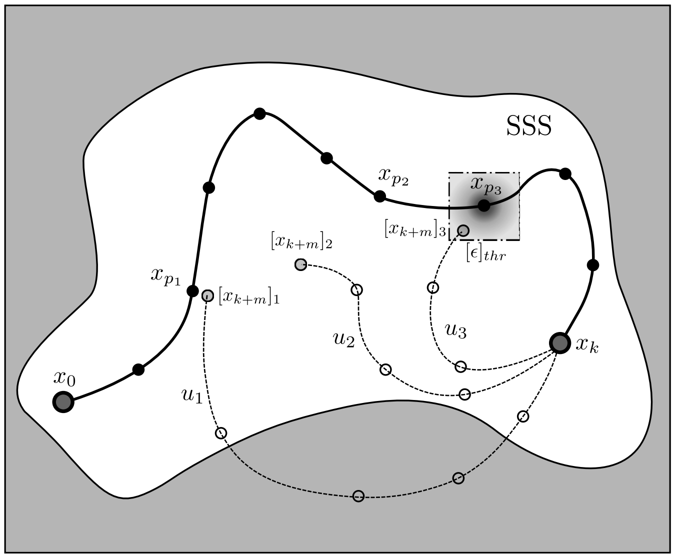


Fig. 2. Example of backup evaluation. Starting from position $\boldsymbol{x}(0) = \boldsymbol{x}(t_0)$, the agent has followed its trajectory to current state $\boldsymbol{x}_k$. The three dashed lines represent the means of three proposed interval backups from $\boldsymbol{x}_k$. The proposed backup generated by control $\boldsymbol{u}_1$ is discarded since it violates the known safe state space SSS. The proposed backup generated by $\boldsymbol{u}_2$ is also discarded since $[\boldsymbol{x}_{k+m}]_2$ does not satisfy the closeness condition with nearest explored state $\boldsymbol{x}_{p_2}$. Control $\boldsymbol{u}_3$ generates a feasible backup, since the associated trajectory lies entirely in the SSS, and since $[\boldsymbol{x}_{k+m}]_3$ respects the closeness condition.

This definition replaces the reach with intervals $[\boldsymbol{\epsilon}]$ and $[\boldsymbol{\delta}]$. Given two symmetric threshold intervals $[\boldsymbol{\epsilon}]_{\text{thr}}$ and $[\boldsymbol{\delta}]_{\text{thr}}$, a backup is effective if either $[\boldsymbol{\epsilon}] \subset [\boldsymbol{\epsilon}]_{\text{thr}}$ or $[\boldsymbol{\delta}] \subset [\boldsymbol{\delta}]_{\text{thr}}$. Fig. 2 shows an example of backup evaluation. Length m should be chosen so that the size of intervals $[\boldsymbol{x}_{k+m}]$ is comparable to risk perception ranges ϵ.

The choice of the threshold intervals is empirical in nature; however $[\boldsymbol{\epsilon}]_{\text{thr}}$ should depend on the magnitude of the control. Given assumption 4, $\lim_{\epsilon \to 0} \max_k \|\boldsymbol{u}(t_k) - \boldsymbol{u}^*(t_k)\| = 0$, so that the smaller the threshold $[\boldsymbol{\epsilon}]_{\text{thr}}$, the more likely the feasibility of control $\boldsymbol{u}(t_k)$. To account for this, an initial threshold interval $[\boldsymbol{\epsilon}]_{\max}$ is assigned to each new state $\boldsymbol{x}_k$. At each successive time step, all the threshold intervals are shrunk by a factor η

$$\eta = \left(\prod_{i=1}^{m} \eta_i\right)^{\frac{1}{m}}; \quad \eta_i = \left(1 - \frac{\left|u_i(t_k) - \frac{1}{2} \cdot (\underline{u}_i + \overline{u}_i)\right|}{\frac{1}{2} \cdot (\overline{u}_i - \underline{u}_i)}\right)^{\lambda} \tag{11}$$

with positive exponent $\lambda < 1$. The more the control action nears the boundaries $\underline{\boldsymbol{u}}$ and $\overline{\boldsymbol{u}}$, the smaller the η, and the more the shrinkage of $[\boldsymbol{\epsilon}]_{\text{thr}}$. Exponent λ also regulates shrinkage: the smaller λ, the less previous explored states are penalized with respect to recent ones. Values of λ and of $[\boldsymbol{\epsilon}]_{\max}$ are closely related, e.g., a small λ should be paired with a small $[\boldsymbol{\epsilon}]_{\max}$ and a high λ with an increased $[\boldsymbol{\epsilon}]_{\max}$.

D. SHERPA

In this section, the SHERPA algorithm (Fig. 3) will be discussed in detail. At the start of the exploration, the agent is in state $\boldsymbol{x}_0$. It is assumed that $W(\boldsymbol{x}_0) = 0$. For each proposed action $\boldsymbol{u}_p$ generated by policy π, SHERPA computes

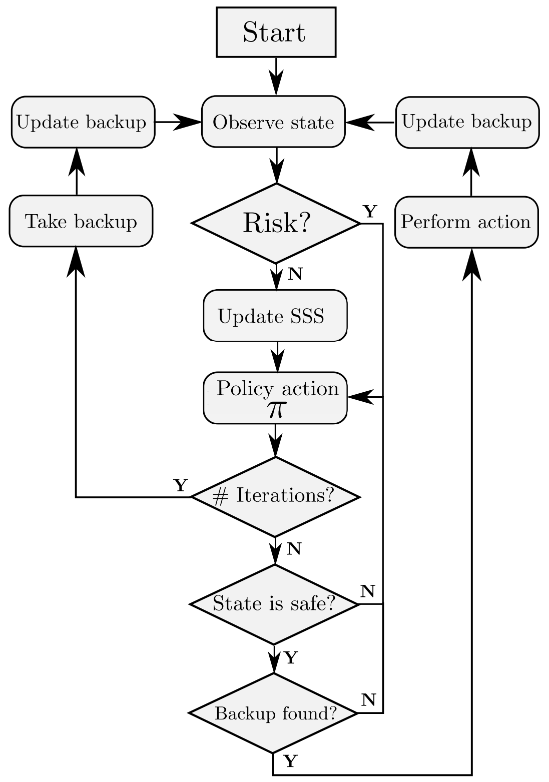


Fig. 3. Flowchart summarizing the procedure of SHERPA. The policy action is checked for safety and for backups; if the checks succeed, the action is taken and the new backup is stored in place of the previous. If the iteration limit is reached without success, SHERPA recollects the previously stored backup.

the predicted interval $[\boldsymbol{x}_p] = \boldsymbol{\Delta}(\boldsymbol{x}_0, \boldsymbol{u}_p)$. If $[\boldsymbol{x}_p] \notin \text{SSS}(t_0)$, a different action is proposed, or the agent might encounter a fatal state. Otherwise, SHERPA checks the existence of a backup for $[\boldsymbol{x}_p]$: a random finite control sequence $\boldsymbol{u}_b$ is generated, and for each step, its reach is checked as per Definition 10. If no backup is found after a set number of iterations, the current action is discarded, and a different action is proposed. In the new state $\boldsymbol{x}_1$, if $W(\boldsymbol{x}_1) = 0$, then SSS is augmented with all the states currently in reach of the risk perception. The procedure is iterated during the whole exploration. In theory, the agent has always a feasible control at its disposal, eventually resorting to a backup, i.e., an iterative control, or a dynamical equilibrium. If all proposed actions fail the previous checks, SHERPA will force the agent to adopt the current backup. During backup execution, it will proceed to individuate a safe control and a backup for the expected arrival interval. The complexity of the algorithm depends on the number of visited states k, the number of available actions n_a, the maximum amount of iterations for action selections $a_{\max}$ and for backup checking $b_{\max}$, and is equal to $O(a_{\max} \cdot \max(\min(b_{\max}, n_a^m), kn^k))$.

Other methods have proven their worth in preventing a controlled system from reaching a set of unsafe states or configurations. Potential fields [40] were successfully adapted for robotic navigation and path planning with lack of environmental knowledge, e.g., uncertainty in observed obstacles [41], in the number of obstacles [42] and even in the entire environment [43]. However, these methods are not equally adaptable when the uncertainty extends to the model dynamics, and when holonomic and control constraints are considered, as in this paper. An application of this is provided in the following as comparison with SHERPA. Lyapunov barrier functions [44], control barrier functions [45], and control Lyapunov-barrier functions [46] can provide control that is guaranteed to be safe. However, the underlying assumptions on system structure, or on the barrier functions, can limit their applicability, especially when considering model uncertainty.

E. Quadrotor Task

In this section, a task is simulated for a 2-D quadrotor flying in a room. The underlying RL agent adopts a fully random policy to explore. The goal is for SHERPA to prevent collision and thus enable safe state exploration. The state vector is $(x,\ \dot{x},\ z,\ \dot{z},\ \text{and}\ \theta)^T$, where x, z and θ are the horizontal and vertical coordinate of the quadrotor, and the pitch angle. The bounding model[2] is

$$\begin{aligned}\ddot{x} &= -\frac{[c]}{[m]}|\sin\theta||\dot{x}|\dot{x} + \frac{T}{[m]}\sin\theta; \quad \dot{\theta} = [k]I_\tau \\ \ddot{z} &= -\frac{[c]}{[m]}|\cos\theta||\dot{z}|\dot{z} + \frac{T}{[m]}\cos\theta - g \end{aligned} \tag{12}$$

where c and $[c]$ is a damping coefficient; m and $[m]$ is the mass of the quadrotor; T is the total thrust generated by the rotors; k and $[k]$ is an efficiency factor for torque impulse I_τ; g is the gravitational acceleration. Intervals $[c]$, $[m]$, and $[k]$ are

$$[m] = [0.344, 0.516]\ \text{kg}; \quad [c] = [0.72, 1.08] \times 10^{-5}\ \frac{\text{kg}}{\text{m}}$$
$$[k] = [0.8, 1.2]$$

with control action bounded as $T \in [0, 12.7]$ N and $I_\tau \in [-300°, 300°]\ \text{s}^{-1}$. The model of the quadrotor used for the computation of the real dynamics is obtained from the bounding model by randomly selecting c, m and k from their intervals.

The quadrotor initially hovers at the middle of a room three meters wide and two meters high. Colliding with the walls or with the floor/ceiling is a fatal occurrence, and the quadrotor perceives risk half a meter from the ceiling or floor, or at one meter from a wall. The reach of the backup depends on

$$[\boldsymbol{\epsilon}^{\max}] = \left(\left[-\frac{1}{2}, \frac{1}{2}\right]\ \text{m}; \left[-\frac{1}{2}, \frac{1}{2}\right]\ \frac{\text{m}}{\text{s}}; \left[-\frac{1}{2}, \frac{1}{2}\right]\ \text{m}; \left[-\frac{1}{2}, \frac{1}{2}\right]\ \frac{\text{m}}{\text{s}}; \left[-\frac{\pi}{6}, \frac{\pi}{6}\right]\right).$$

During exploration, control is bounded between 10% and 90% of the original intervals: $T \in [1.3, 10.3]$ N and $I_\tau \in [-270°, 270°]\ \text{s}^{-1}$: this prevents factor η of (11) to vanish. Coefficient λ is chosen as 0.5. As for $[\boldsymbol{\delta}]$, taking into consideration the dynamics in (12), it is evident that the position (x, y) of the quadrotor itself does not influence the value of the derivatives. Therefore, the interval $[\delta_{\text{thr}}] = ([-\infty, \infty]; [-1, 1]\ \text{m/s}; [-\infty, \infty]; [-1, 1]\ \text{m/s}; [-45, 45]°)$

[2]It can be shown that, due to local Lipschitz continuity of (12), Assumption 4 is valid for any realization of the dynamics.

with equilibrium point $\dot{x} = 0$, $\dot{z} = 0$, $\theta = 0$ is selected. SHERPA evaluates up to ten actions per time step, proposed by the random policy. A maximum of 40 backup evaluations are performed, for a maximum of 400 iterations per time step. Initially, since the agent does not perceive any risk, SSS(t_0) coincides with

$$\left([-1, 1]\ \text{m}; [-\infty, \infty]; \left[-\frac{1}{2}, \frac{1}{2}\right]\ \text{m}; [-\infty, \infty]; [-\infty, \infty]\right)$$

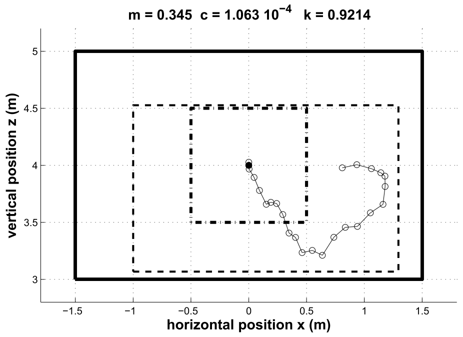
and it is updated whenever $W = 0$ as

$$\text{SSS} \cup \left\{\mathcal{S}(x', z', \dot{x}, \dot{z}, \theta) \| x(t) - x'| < 1 \vee |z(t) - z'| < \frac{1}{2}\right\}.$$

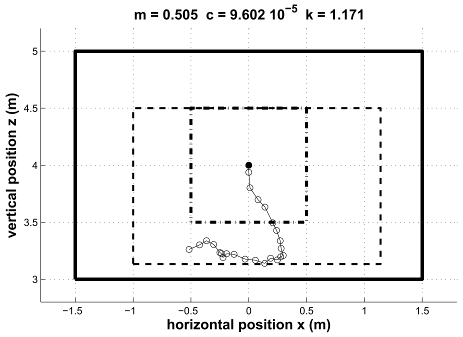
A series of 4 runs of SHERPA is shown in Fig. 4. The values of c, m and k are indicated in the captions. The dots indicate the position of the quadrotor at different time steps, starting in position (0, 4). The solid rectangle represents the fatal states. The dash-dotted line represents the contour of the region where no risk is individuated ($W(x, y) = 0$) and the SSS is therefore updated. The dashed line represents the known SSS at the end of the run. In 3 out of 4 cases, the quadrotor reaches the boundaries of the SSS, but manages to safely divert its trajectory [Fig. 4(a) and (b)] or to wait in position until another path has been found [Fig. 4(c)].

In general, the agent chooses actions that result in safe trajectories, even with a significant excursion in the value of parameters. Nonetheless, the random nature of the underlying policy is still noticeable. If a different task were to be proposed (e.g., waypoint navigation), SHERPA could still be utilized to evaluate actions suggested by a nonexploratory, goal-oriented policy.

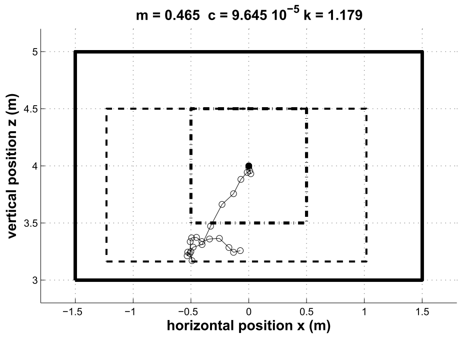
In order to highlight the novelty of the proposed approach, Figs. 5 and 6 show an application of a different method, specifically the basic potential field (BPF) method [47] for collision avoidance. The BPF differs from other potential field methods such as [40] in that it generates a finite potential accounting for velocities and maximum decelerations. In this application, the BPF repels the quadrotor from exiting the known SSS. The rate at which the field reaches its maximum intensity is given by its gain G and by the worst case maximum deceleration as given by the uncertain model. The quadrotor in Fig. 5 is in an almost nominal condition of mass and torque effectiveness [as in Fig. 4(d)]. It can be seen how a BPF with gain $G = 0.5$ prevents collisions. The trajectory is centered around the middle of the known SSS, where the combined repulsive field, averaged between the different velocities, has a minimum. This differs from the proposed approach, in which the outer region of the SSS is not penalized with respect to its interior [see Fig. 4(a)]. Fig. 6 shows the resulting trajectory when applying the previous BPF to a quadrotor with increased mass and torque effectiveness, as in Fig. 4(b). The figure shows how the quadrotor flies predominantly in the lower half of the room, as a result of the increased mass of the quadrotor. The trajectory violates the SSS, and even ends with a collision. This is due to the following. First, the quadrotor is forced to continuously change its attitude so that resultant of the forces complies with the direction indicated by the BPF. This essentially turns the BPF into a reactive collision avoidance method. As a result of this, the effectiveness of the method depends on the gain G, whose value determines the reactivity of the field, as shown by the



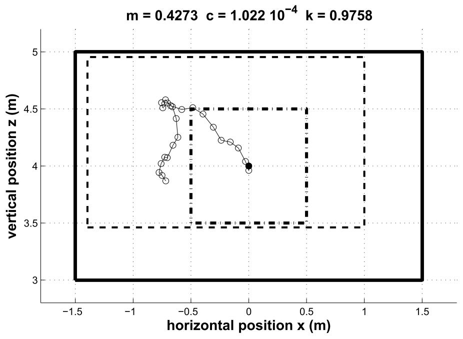
(a)



(b)



(c)



(d)

Fig. 4. Dots represent the quadrotor position in time. The solid rectangle represents the fatal states that the agent must avoid. The dotted and dashed line delimits the states where the agent does not perceive risk. The dashed line represents the known SSS at the end of each run. (a) Run with $m = 345$ g, $c = 1.06 \times 10^{-4}$ kg/m, and $k = 0.92$. (b) Run with $m = 505$ g, $c = 9.60 \times 10^{-5}$ kg/m, and $k = 1.17$. (c) Run with $m = 465$ g, $c = 9.64 \times 10^{-5}$ kg/m, and $k = 1.18$.

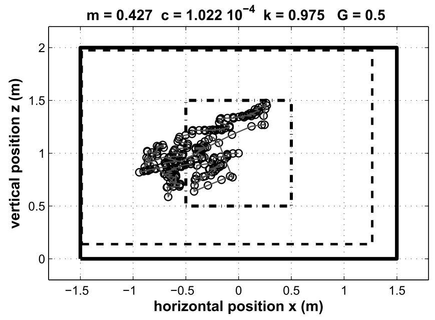


Fig. 5. Trajectory obtained when using the BPF to control a quadrotor in an almost nominal condition of mass and torque efficiency.

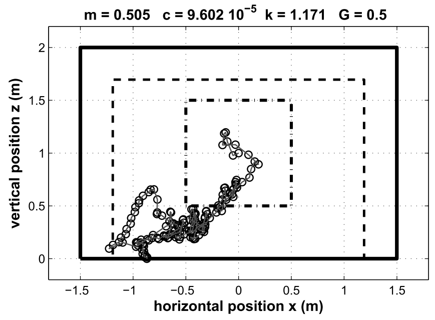


Fig. 6. Trajectory obtained when using the BPF to control a quadrotor with increased mass and torque efficiency. The quadrotor flies predominantly in the lower half of the room before colliding with the floor.

two different conditions of Figs. 5 and 6. If G is too low, the BPF is not sufficiently reactive; however, too high a gain can cause instability and unwanted oscillations. This selection of G is especially critical when considering systems with model uncertainties. This constitutes another difference with SHERPA, which takes into account all possible realizations of the model due to its interval formulation.

IV. OptiSHERPA

A. Motivation and Algorithm Description

Sec. III introduced SHERPA, which relies on an *a priori* defined interval to define closeness. This approach was shown to succeed in the quadrotor application. However, this approach has two drawbacks. First, if the system is not easily controllable, predicted arrival intervals might fail in satisfying the closeness condition. For these systems, a safe backup could involve a complex trajectory, which would require a considerable amount of time to be followed. The trajectory might also reach a portion of the state space outside of the risk-perception range of SHERPA, thus requiring a certain degree of "global" knowledge to be ensured as safe. The confidence in state prediction and the current knowledge of the SSS (provided by the risk-perception) might not be sufficient to identify such a trajectory as a backup. This would lead to situations where SHERPA is either unable to find a suitable control, thus exhausting its backup and then rejecting all further actions, or is forced to keep the system on hold in an equilibrium point.

A second drawback of the closeness condition is that, acting as a yes-or-no filter, it does not distinguish between control actions that are almost satisfactory and those that are completely unacceptable. After resorting to a backup, and while trying to find a new one from the arrival condition, SHERPA has only a limited amount of iterations to find a new backup. In the event that this search is unsuccessful, SHERPA is forced to take a possibly unsafe action.

For the above reasons, a second version of SHERPA named OptiSHERPA is presented. It differs from SHERPA in two ways. First, OptiSHERPA introduces metrics for the selection of actions. A finite set of actions is evaluated at each time step together with a feasible backup, and the best action is performed. The introduction of metrics reduces the burden of online application by allowing OptiSHERPA to rank its options and to take an informed decision if the available amount of online iterations is depleted. A second difference lies in the strategy itself. With SHERPA, the generation of backups would keep the system safe, providing a possible escape route at every time step. With OptiSHERPA, when the agent does not perceive danger, a *distance metric* is implemented with the goal of preventing potentially unsafe behavior of the system, effectively constraining the dynamics similar to SHERPA. When danger is detected, the agent examines its current belief in the state space and actively avoids the regions of the state space that are least safe. This is done via an *evasion metric*.

The remainder of this section is as follows. First, the metrics will be discussed in detail. Second, the algorithm will be illustrated. Third, a simulated application to an elevator control task will be shown.

B. Metrics

1) Distance Metric: This metric allows to classify intervals based on distance so that, during the evaluation of backups, the one with "closest" reach can be selected. Indicating by $\odot$ the Schur product

$$d(\boldsymbol{x}, \boldsymbol{I}) = \left\| \left(\boldsymbol{x} - \frac{\overline{\boldsymbol{I}} + \underline{\boldsymbol{I}}}{2} \right) \odot \boldsymbol{v}_r \right\| + \rho \cdot \left\| \left(\frac{\overline{\boldsymbol{I}} - \underline{\boldsymbol{I}}}{2} \right) \odot \boldsymbol{v}_r \right\| \tag{13}$$

is a "distance"[3] between $\boldsymbol{x}$ and the center of the interval, rescaled by $\boldsymbol{v}_r \in \mathbb{R}^{n+}$, plus a term proportional to the interval width weighted by a positive parameter $\rho < 1$. This term allows to include the uncertainty in the interval as a penalizing factor. A lower value of ρ privileges intervals whose center are nearer regardless of their width; conversely a higher value penalizes intervals whose center is nearer but whose elements are more dispersed. Fig. 7 shows this transition. Vector $\boldsymbol{v}_r$ should be chosen so that, for any two excursions in state $\Delta\boldsymbol{x}^{\mathrm{I}}$ and $\Delta\boldsymbol{x}^{\mathrm{II}}$, it is $\|\Delta\boldsymbol{x}^{\mathrm{I}} \odot \boldsymbol{v}_r\| > \|\Delta\boldsymbol{x}^{\mathrm{II}} \odot \boldsymbol{v}_r\|$ if and only if excursion $\Delta\boldsymbol{x}^{\mathrm{I}}$ affects safety and controllability of the system more than excursion $\Delta\boldsymbol{x}^{\mathrm{II}}$. Thus, to components x_j related to risks should correspond an adequately big vector component v_j. A second function of $\boldsymbol{v}_r$ is to normalize distances in $\mathcal{S}$, since components x_i might have different units of measure. The magnitude of the control can also be accounted for by

[3]Equation (13) is not rigorously a distance, as it is not defined on $\mathbb{R}^n \times \mathbb{R}^n$ but on $\mathbb{R}^n \times \mathbb{I}^n$, where $\mathbb{I}^n$ is the set of n-dimensional intervals. However, if (13) is restricted to the subset of "crisp" intervals, it is analog to the Euclidean distance in $\mathbb{R}^n$, hence the denomination.

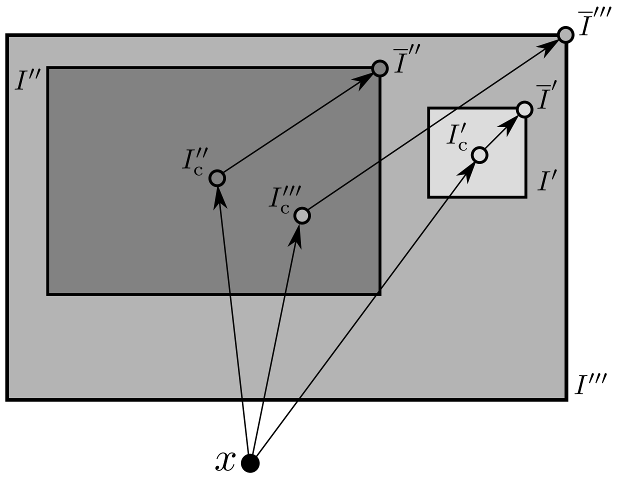


Fig. 7. Choice of ρ affects the outcome of the metric when comparing different intervals. For lower values of ρ, interval I''' is the nearest, since its center I'''_c is the nearest to x. By gradually increasing ρ, 13 will select interval I''. Finally, for ρ approaching 1, I' will be the nearest.

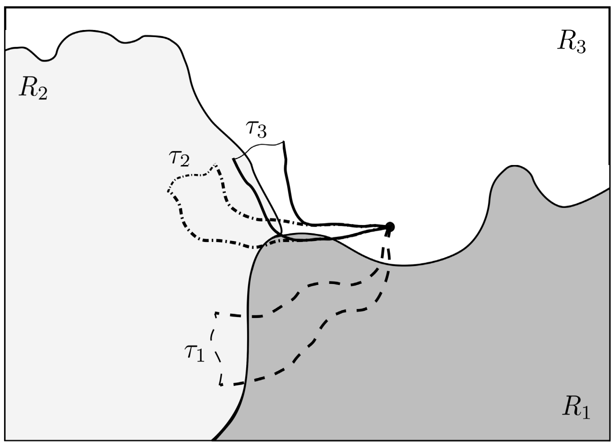


Fig. 8. $\mathcal{S}$ comprises fatal region R_1, unknown region R_2, and safe region R_3. All τ have a non-empty intersection with R_1. However, τ_1 crosses R_1 in its entirety and is therefore the least safe; τ_2 and τ_3 also cross the region, but τ_3 has the minimal overlap with both R_1 and R_2, and the biggest overlap with R_3: it is therefore the safest.

a similar method as the one illustrated in (11). At each time step compute η so that the metric distance between $\boldsymbol{x}$ and interval $\boldsymbol{I}(t_i)$ at time at time t_j is

$$d_\eta(\boldsymbol{x}, \boldsymbol{I}(t_i), t_j) = d(\boldsymbol{x}, \boldsymbol{I}(t_i)) / \prod_{k=i}^{j} \eta(t_k) \tag{14}$$

In conclusion, when risk perception and trajectory prediction are suboptimal, the distance metric selects an action whose predicted arrival states differ the least from a known visited state.

2) Evasion Metric: The evasion metric evaluates trajectories based on the current belief in the composition of the state space. Consider for example a state space partitioned in a safe region R_{safe}, a fatal region R_{fat}, and an uncertain region R_{unc}. More regions could be defined as long as they can be ordered from the safest to the most fatal. Each control action sequence generates bounding trajectories τ, which OptiSHERPA computes as a succession of intervals in time. First, among all bounding trajectories, discard those that at any given time will entirely be comprised in a fatal region: for OptiSHERPA, this is equivalent to generating one interval entirely composed of fatal states. Second, discard those trajectories that do not end up in the safe region at the end of the trajectory, unless no such trajectory exists. The remaining trajectories are those that have the most probability of both avoiding fatal occurrences and of restoring safety. Third, consider how the trajectories overlap the regions, e.g., R_{safe}, R_{fat}, R_{unc}. Assuming that all trajectories in the bounding trajectory are equally likely to occur, the bigger volume of the intersection with a fatal region, the higher the chance of a fatal occurrence when following the trajectory. Consider then two trajectories with similarly sized intersection with the fatal region, but one of which has a higher intersection with the uncertain region than the other. Since part of the uncertain states could actually be fatal, it is preferable to follow the second trajectory, reducing such a risk. This procedure can be summarized as follows.

1) Define a hierarchy of regions R_i where R_i has a higher probability of containing fatal states than R_j, $i < j$.
2) Assign a weight $w_i > 0$ to each region, so that if $i < j$, $w_i > w_j$.
3) Remove from the set of feasible bounding trajectories $\boldsymbol{\tau}$ those τ that intersect a fatal region entirely.
4) Remove from $\boldsymbol{\tau}$ those τ that do not end up in the SSS, unless this depletes τ.
5) For each $\tau_j \in \boldsymbol{\tau}$ compute the volume of the intersection ρ_{ij} with region R_i.
6) The optimal control sequence u^* is the one generating $\tau^* = \text{argmin}_{\tau_j \in \boldsymbol{\tau}} \sum_{R_i} \rho_{ij} \cdot w_i$.
7) Apply the first element of u^*.
8) If $W = 1$, go to 1; otherwise exit.

Fig. 8 provides an example of the above procedure. The result of applying the evasion metric is that the agent will reach for a safe region where danger is no longer perceived, while minimizing the probability of encountering a fatal occurrence.

C. OptiSHERPA

Fig. 9 shows schematically the implementation of OptiSHERPA. At the start of the exploration, the agent is in state $\boldsymbol{x}_0$. It will be assumed that $W(\boldsymbol{x}_0) = 0$. OptiSHERPA generates an array of control sequences, including u^π given by the policy. For each sequence, the algorithm checks if the corresponding trajectory τ is included in the SSS. Those τ for which this is not true are removed from the array; the rest are evaluated with the distance metric. The control u corresponding to the optimal τ is selected, and the first action of u^* is then applied. In the new state $\boldsymbol{x}_1$, if $W(\boldsymbol{x}_1) = 0$, the SSS is augmented with all the states currently in reach of the risk perception, and the process is repeated. If $W(\boldsymbol{x}_1) \neq 0$, the SSS is not modified, and the evasion metric is implemented. OptiSHERPA generates an array of control sequences and corresponding τ as before. According to the procedure of Section IV-B2, an optimal control sequence u^* is found, and its first action is implemented. The evasion metric is

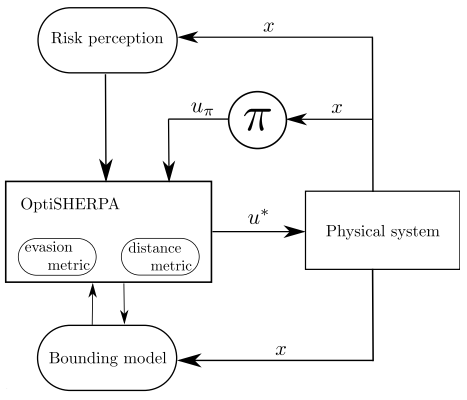


Fig. 9. Architecture of OptiSHERPA. At each time step, the algorithm receives information in the form of current risk perception and suggested policy control u_π. The bounding model is used to predicted trajectories, which in turn are evaluated by implementing either the distance or the evasion metric depending on current risk perception. Then, the first action of u^* is taken, which generates a new state.

used until $W(\boldsymbol{x}(t)) = 0$, after which the distance metric is reimplemented. This procedure is followed until the end of exploration.

D. Elevator Control Task

1) Model: The goal of the application is to simulate online training for an RL agent on-board of a fighter aircraft, which is constrained to fly in a range of ± 40 ft from starting height. It is assumed that a stall occurs if the angle of attack $\alpha < -15^\circ$ or $\alpha > 12^\circ$. Violating this envelope causes a fatal occurrence in this environment. A bounding model of the aircraft is available as in (15) and (16)

$$\dot{h} = V \cdot \sin(\theta - \alpha) \tag{15}$$

$$\begin{pmatrix} \dot{\theta} \\ \dot{\alpha} \\ \dot{q} \end{pmatrix} = \begin{bmatrix} 0 & 0 & 1 \\ 0 & c_{\alpha\alpha} & c_{\alpha q} \\ 0 & c_{q\alpha} & c_{qq} \end{bmatrix} \cdot \begin{pmatrix} \theta \\ \alpha \\ q \end{pmatrix} + \begin{bmatrix} 0 \\ c_{\alpha\delta_e} \\ c_{q\delta_e} \end{bmatrix} \cdot (\delta_e + \Delta\delta_e) + \begin{pmatrix} 0 \\ 0 \\ \Delta\dot{q} \end{pmatrix} \tag{16}$$

with

$$V \in [450 \quad 550]\, \frac{\text{ft}}{\text{s}}; \quad c_{\alpha\alpha} \in [-0.70 \quad -0.58]$$
$$c_{\alpha q} \in [0.76 \quad 0.95]; \quad c_{q\alpha} \in [-1.72 \quad -1.41]$$
$$c_{qq} \in [-0.97 \quad -0.79]; \quad \Delta\delta_e \in [-0.15^\circ \quad 0.15^\circ]$$
$$\Delta\dot{q} \in [-0.5 \quad 0.5] \quad c_{\alpha\delta_e} = -1.4 \cdot 10^{-3}; \quad c_{q\delta_e} = -0.1137$$

where V [ft/s] is the constant flight speed, height h [ft] is the change in height, θ [rad], α [rad] are changes of pitch angle and angle of attack, and q [rad/s] is the pitch rate. Vector $(h, \theta, \alpha, q)^T$ is the state of the system and δ_e [deg] is the elevator deflection.

The intervals of (15) and (16) are obtained as follows. First, the nonlinear dynamics are linearised at 15 000 ft of height and 500 ft/s of speed. Then, resulting coefficients $c_{\alpha\alpha}$, $c_{q\alpha}$ and c_{qq}, as well as speed V, are altered by $\pm 10\%$, whereas $c_{\alpha q}$ is reduced between 80% and 100% of the original

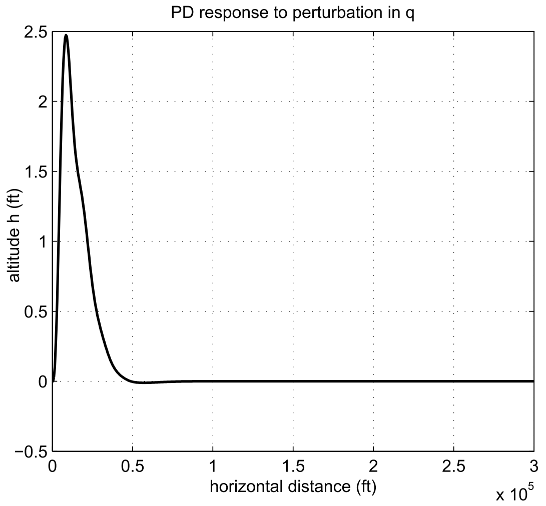


Fig. 10. Change in flight height of the system equipped with the PD controller after a perturbation in q. The final error in height is brought to zero.

value. $\Delta\dot{q}$ represents uncertainty in pitch dynamics, and $\Delta\delta_e$ represents an error in effective deflection. Finally, terms $c_{\alpha\delta_e}$ and $c_{q\delta_e}$ are crisp elevator coefficients. Among all possible representations of the dynamics, the one with coefficients

$$V = 550\frac{\text{ft}}{\text{s}}; \quad c_{\alpha\alpha} = -0.58; \quad c_{\alpha q} = 0.83$$
$$c_{q\alpha} = -1.586; \quad c_{qq} = -0.97; \quad \Delta\dot{q} = -0.5$$

is selected: cross terms $c_{q\alpha}$ and $c_{\alpha q}$ as the mean of their interval, and remaining terms as one of their extrema. The deflection error $\Delta\delta_e$ is treated as noise.

A PD controller provides a baseline policy for the RL agent. Deflection δ_{PD} is the sum of a proportional and derivative term in γ, of a proportional term in height h, and a proportional damping term in pitch rate

$$\bar{\delta}_{\text{PD}} = \frac{1}{200} \cdot h + \frac{1}{100} \cdot \dot{h} + 10 \cdot (\theta - \alpha) + 4 \cdot (\dot{\theta} - \dot{\alpha}) + \frac{1}{2} \cdot q$$
$$\delta_{\text{PD}} = \max(\min(\bar{\delta}_{\text{PD}}, 1^\circ), -1^\circ).$$

The controller is hand tuned to achieve a satisfactory performance in nominal conditions. Fig. 10 shows a stable and well damped response to a perturbation in q of 0.5°/s. The proportional term in h brings the final error in altitude to zero.

The RL agent follows an ϵ-greedy policy by selecting random actions $\delta \in [-1; 1]$ with probability $\epsilon = 0.2$, and $\delta = \delta_{\text{PD}}$ otherwise.

With the system differing from nominal conditions, and with the addition of random deflections, the agent is not always able to abide to the constraints in height and angle of attack. A dedicated SHERPA could facilitate the task. However, with respect to the simple quadrotor model of the previous section, the aircraft model represents an increased challenge for SHERPA. In particular, consider the height dynamics represented by (15). Change in height is achieved via the flight path angle $\gamma = \theta - \alpha$, and in turn $\dot{\gamma} = \dot{\theta} - \dot{\alpha} \cong 0.062 \cdot q + 0.56 \cdot \alpha$, so that $\dot{\gamma}$ and $\dot{h}$ are influenced by the angle of attack. This represents the direct correlation between lift and flight path angle: α must increase (decrease) in order to increase (decrease) γ. The dynamic in γ is considerably slower than the one in θ, α, and q, and the evolution in h for two different control sequences can be appreciated only after a certain amount of time steps. Meaningful variations in the flight height can be observed only in further-time estimates, which are also the most uncertain. This motivates the use of OptiSHERPA.

The algorithm is initialized with the knowledge of the two risks in h and α, and two corresponding risk perception ranges of 15 ft and 5°, respectively. Whenever OptiSHERPA finds a range of height or angle of attack to be safe, it immediately adds this range to its internal SSS. SSS is then an interval bounded in h and α by the highest and lowest values obtained from risk perception, and unbounded in θ and q, which have no risk.

The agent generates one command sequence: either a full PD predicted sequence, or a random action plus the command predicted by the PD at later steps. Each sequence has a duration of six time steps of 0.2 s. Then, if danger is perceived, or if $|\gamma| > 0.5°$, the agent enables OptiSHERPA. When enabled, the algorithm generates multiple command sequences, each lasting six time steps, in the form $\boldsymbol{u} = (u_0, u_b, \ldots, u_b)$, $u_0, u_b \in \{-0.75°, -0.25°, 0°, 0.25°, 0.75°\}$. Command u_0 is the OptiSHERPA proposed action, and the constant u_b acts as a backup. This formulation of $\boldsymbol{u}$ reduces the total number of metric evaluations from 5^6 to 25. The distance (14) is computed for all the above sequences, with

$$\boldsymbol{v}_r = \left[0.2\ (\text{ft})^{-1}, \frac{36}{\pi}, \frac{36}{\pi}, \frac{4}{\pi}\ \text{s}\right]^T, \quad \rho = 0.5, \quad \lambda = 0.1.$$

If danger is perceived, the evasion metric is implemented, and the state space is partitioned into distinct regions, depending on the risk perception and the knowledge of the SSS. If only one of either RFSS_h or RFSS_α is detected, the following are defined:

1) a "black" region R_{bk} comprising all states with fatal values of h (or α);
2) a "white" region R_{wh} comprising all states within risk perception of the nearest fatal value;
3) a "green" region R_{gn} equal to the SSS $\setminus R_{\text{wh}}$;
4) a "gray" region R_{gy} equal to the remainder of $\mathcal{S}$.

The white region is introduced in order to encourage the metric to move away from those regions where risk is still perceived, thus exiting the evasion cycle. If both h and α are perceived, then two regions are added: a "red" region R_r comprising the states that are fatal for one feature, but not for the other, and a "blue region" R_{bl} comprising those states within risk perception of the nearest fatal value of one feature, but not of the other. Finally, to each region is assigned a weight

$$w_{\text{bk}} = 2, \quad w_r = 0, \quad w_{\text{gy}} = -0.1, \quad w_w = -0.2$$
$$w_{\text{bl}} = -1, \quad w_{\text{gn}} = -2.$$

A total of 500 trials is performed from the starting perturbed condition $h = \alpha = \theta = 0$, $q = 0.5°$. Each trial has a duration of 600 time steps, equivalent to 120 s. In each trial, both the "original" RL agent and the one augmented with SHERPA are run at the same time. The added noise on the elevator deflection and the random action were the same at all time for both the original and the augmented agent, albeit SHERPA is allowed to dismiss the random action as previously exposed.

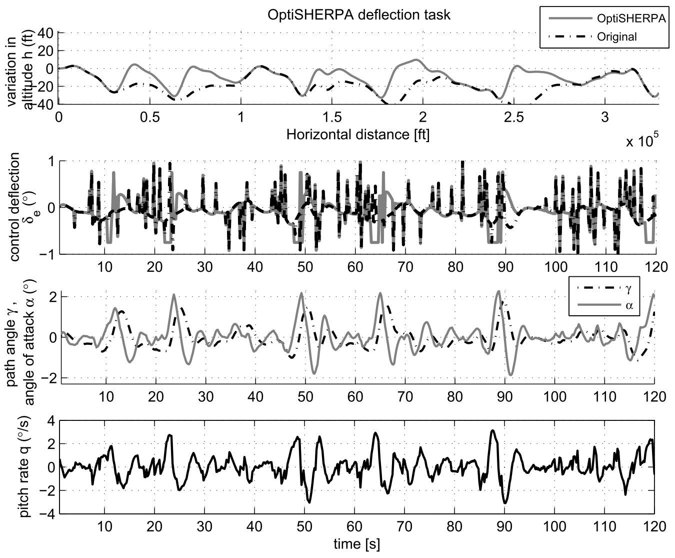


Fig. 11. Top plot represents the trajectory with the original RL agent and with the addition of OptiSHERPA. The second plot represents deflection δ_e for the two different agents; OptiSHERPA selects constant maximal deflections during recovery by optimizing the evasion metric. The third shows the variations of γ and α during flight with OptiSHERPA. The bottom plot shows pitch rate q.

Fig. 11 shows a typical result for the task. The upper plot shows the two trajectories with the application of OptiSHERPA and with the original agent. At the start of the task, the original agent manages to keep deviations of γ, α, and q within reasonable limits, thus satisfying the requirements for OptiSHERPA. However, the effect of OptiSHERPA becomes noticeable when a risk in h is perceived. In all those instances, the algorithm finds a reactive solution in the first iterations, increasing the pitch rate and the angle of attack. Then, in later iterations when recovery is incipient, OptiSHERPA applies a deflection of opposite sign. The reason for this is as follows. Both the flight path angle and the angle of attack have changed sign, so that the bounding model predicts a recovery in flight height in near time. However, the positive pitch angle could increase further the angle of attack. Since such levels have not been experimented before, the evasion metric considers this a potentially threatening situation. Thus, the best course of action for OptiSHERPA is to achieve negative pitch rate to reduce α. Therefore, lift increases during the first iterations, and reduces it after recovery is in progress. As for q, rapid changes are performed during recoveries; when not in recovery, the agent keeps q within acceptable limits. By means of following the evasion metric, OptiSHERPA consistently recovers in a more incisive and fast way than the equivalent PD action, and prevents violations in height during the whole flight. Without this additional recovery capability, the original RL agent violates the constraint in flight height in 53% of the trials. It is necessary to consider, however, that the original controller is always adopting an exploratory ϵ-greedy policy. A direct comparison between the two should therefore be avoided.

The angle of attack plays a crucial role in determining whether a recovery can be achieved with the limited distance

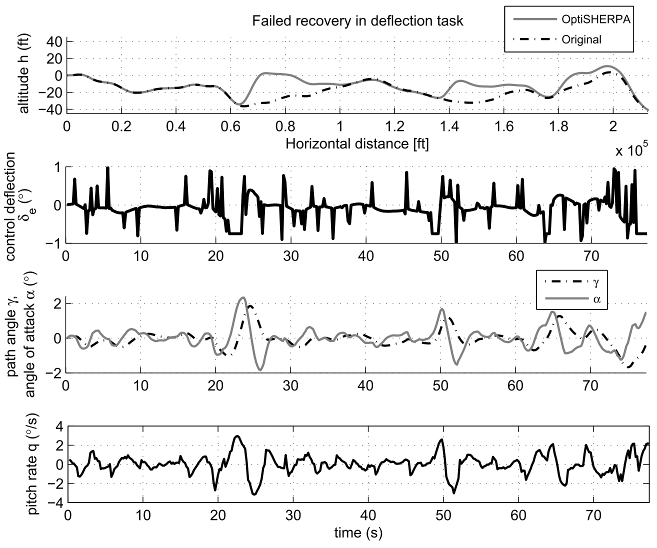


Fig. 12. Typical violation of the flight height constraints.

in feet allowed by the risk perception. Fig. 12 shows one such case. Prior to the violation, a series of positive random deflections reduces considerably the flight path angle while at the same time decreasing the angle of attack and generating negative pitch rate. As soon as the risk is perceived, and OptiSHERPA is engaged with the evasion metric, the agent increases the pitch rate and starts generating additional lift through α, but the violation still occurs. Nonetheless, it can be seen how OptiSHERPA suggest the correct maximum deflection, and how the residual negative γ is reduced in the instants prior to violation. Approximately 10% of the simulations incurred in such a violation for the agent with OptiSHERPA. However, two considerations must be added. First, all violations occurred following cumulative random deflections, giving rise to conditions from which recovery was very difficult even with maximum deflection, and resulted in minor violations (as in Fig. 12). Second, all cases where violation occurred for OptiSHERPA were either concurrent or preceded by a violation for the original RL agent.

OptiSHERPA showed the correct behavior in case of imminent flight height violation, selecting strong deflections in the first iterations and subsequently reducing the variation of angle of attack introduced. This resulted in quick and safe recoveries. For a fraction of the runs, the stochasticity in the agent action gave rise to small flight height violations, although without any reduction in performance when compared to the alternative original RL agent, which was equally affected.

V. Conclusion

This paper presents a new approach for autonomous agents in dangerous environments. The presence of fatal and LTF states constitutes the motivation for the approach, with risk perception as its main assumption. The notions of safe control and backup are introduced to present an algorithm for safe exploration: SHERPA. By relying on a bounding model of the dynamics, SHERPA allows those policy actions for which the system can be brought near a previously known state by satisfying a closeness condition, thus promoting safety. SHERPA is tested in one simulated quadrotor application, achieving safety. Subsequently, OptiSHERPA is introduced in order to handle tasks for which either the risk perception or the bounding model were insufficient for the original SHERPA. These limitations are addressed by adding metrics, which provide the agent with informed options, and by explicitly including an evasion strategy in those cases where danger is imminent. OptiSHERPA is tested on a second simulated task: maintaining straight flight for a fighter aircraft exploring a strict envelope, with the addition of noise, uncertain dynamics and random exploratory actions. The application shows how the resulting RL agent with OptiSHERPA manages reliable control, adopting a very reasonable behavior during recoveries. The proposed approach constitutes a significant effort into tackling the exploration problem for RL agents on a general level, while not focusing on a particular category of tasks. This is reflected in the absence, within the algorithm, of a high-fidelity model for exploration. This strategy is therefore in line with the model-free approach of RL and of adaptive controllers in general. Future development will include an investigation of: risk perception for real-life scenarios by using sensor-information, methods for the autonomous and possibly adaptive selection of the open parameters of the algorithms (such as m and v_r) and representations of uncertainty not based on interval methods to reduce computational complexity.

References

[1] H. Kwakernaak, "Robust control and h∞-optimization—Tutorial paper," *Automatica*, vol. 29, no. 2, pp. 255–273, 1993. [Online]. Available: http://doc.utwente.nl/29962/

[2] E. van Kampen, Q. P. Chu, and J. A. Mulder, "Continuous adaptive critic flight control aided with approximated plant dynamics," in *Proc. AIAA Guidance, Navigat. Control Conf. Exhibit*, Keystone, CO, USA, Aug 2006, pp. 6429-6456.

[3] S. Ferrari and R. F. Stengel, "Online adaptive critic flight control," *J. Guid. Control Dyn.*, vol. 27, no. 5, pp. 777–786, 2004.

[4] R. S. Sutton and A. G. Barto, *Introduction to Reinforcement Learning*, 1st ed. Cambridge, MA, USA: MIT Press, 1998.

[5] D. Haussler, "Probably approximately correct learning," in *Proc. 8th Nat. Conf. Artif. Intell.*, vol. 2. 1990, pp. 1101–1108.

[6] D. Zhao and Y. Zhu, "MEC—A near-optimal online reinforcement learning algorithm for continuous deterministic systems," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 26, no. 2, pp. 346–356, Feb. 2015.

[7] D. P. Bertsekas and J. N. Tsitsiklis, *Neuro-Dynamic Programming*, 1st ed. Belmont, MA, USA: Athena Scientific, 1996.

[8] J. Si and Y.-T. Wang, "Online learning control by association and reinforcement," *IEEE Trans. Neural Netw.*, vol. 12, no. 2, pp. 264–276, Mar. 2001.

[9] R. Bellman, *Dynamic Programming*, 1st ed. Princeton, NJ, USA: Princeton Univ. Press, 1957.

[10] M. C. Choy, D. Srinivasan, and R. L. Cheu, "Neural networks for continuous online learning and control," *IEEE Trans. Neural Netw.*, vol. 17, no. 6, pp. 1511–1531, Nov. 2006.

[11] Y.-J. Liu, L. Tang, S.-C. Tong, C. L. P. Chen, and D.-J. Li, "Reinforcement learning design-based adaptive tracking control with less learning parameters for nonlinear discrete-time MIMO systems," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 26, no. 1, pp. 165–176, Jan. 2015.

[12] Y.-J. Liu, Y. Gao, S. Tong, and Y. Li, "Fuzzy approximation-based adaptive backstepping optimal control for a class of nonlinear discrete-time systems with dead-zone," *IEEE Trans. Fuzzy Syst.*, vol. 24, no. 1, pp. 16–28, Feb. 2016.

[13] D. Liu, X. Yang, D. Wang, and Q. Wei, "Reinforcement-learning-based robust controller design for continuous-time uncertain nonlinear systems subject to input constraints," *IEEE Trans. Cybern.*, vol. 45, no. 7, pp. 1372–1385, Jul. 2015.

[14] A. Arleo, F. Smeraldi, and W. Gerstner, "Cognitive navigation based on nonuniform Gabor space sampling, unsupervised growing networks, and reinforcement learning," *IEEE Trans. Neural Netw.*, vol. 15, no. 3, pp. 639–652, May 2004.

[15] C. Wang, Y. Li, S. S. Ge, and T. H. Lee, "Optimal critic learning for robot control in time-varying environments," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 26, no. 10, pp. 2301–2310, Oct. 2015.

[16] T. Shimizu, R. Saegusa, S. Ikemoto, H. Ishiguro, and G. Metta, "Robust sensorimotor representation to physical interaction changes in humanoid motion learning," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 26, no. 5, pp. 1035–1047, May 2015.

[17] M. Pecka and T. Svoboda, "Safe exploration techniques for reinforcement learning-an overview," in *Proc. Int. Workshop Modelling Simulation Autonomous Syst.*, Rome, Italy, May 2014, pp. 357–375.

[18] M. Heger, "Consideration of risk in reinforcement learning," in *Proc. 11th Int. Conf. Mach. Learn.*, 1994, pp. 105–111.

[19] J. García and F. Fernández, "A comprehensive survey on safe reinforcement learning," *J. Mach. Learn. Res.*, vol. 16, no. 1, pp. 1437–1480, 2015.

[20] P. Thomas, G. Theocharous, and M. Ghavamzadeh, "High confidence policy improvement," in *Proc. 32nd Int. Conf. Mach. Learn. (ICML)*, 2015, pp. 2380–2388.

[21] R. Neuneier and O. Mihatsch, "Risk sensitive reinforcement learning," in *Proc. Adv. Neural Inf. Process. Syst.*, 1999, pp. 1031–1037.

[22] O. Mihatsch and R. Neuneier, "Risk-sensitive reinforcement learning," *Mach. Learn.*, vol. 49, no. 2, pp. 267–290, 2014.

[23] Y. Shen, M. J. Tobia, T. Sommer, and K. Obermayer, "Risk-sensitive reinforcement learning," *Neural Comput.*, vol. 26, no. 7, pp. 1298–1328, 2014.

[24] P. Geibel and F. Wysotzki, "Risk-sensitive reinforcement learning applied to control under constraints," *J. Artif. Intell. Res.*, vol. 24, pp. 81–108, Jul. 2005.

[25] S. P. Coraluppi and S. I. Marcus, "Risk-sensitive and minimax control of discrete-time, finite-state Markov decision processes," *Automatica*, vol. 35, no. 2, pp. 301–309, 1999. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0005109898001538

[26] K. Driessens and S. Džeroski, "Integrating guidance into relational reinforcement learning," *Mach. Learn.*, vol. 57, no. 3, pp. 271–304, 2004.

[27] B. D. Argall, S. Chernova, M. Veloso, and B. Browning, "A survey of robot learning from demonstration," *Robot. Auto. Syst.*, vol. 57, no. 5, pp. 469–483, 2009.

[28] D. Martinez, G. Alenya, and C. Torras, "Safe robot execution in model-based reinforcement learning," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst. (IROS)*, Sep. 2015, pp. 6422–6427.

[29] C. Gehring and D. Precup, "Smart exploration in reinforcement learning using absolute temporal difference errors," in *Proc. Int. Conf. Auto. Agents Multi-agent Syst.*, 2013, pp. 1037–1044.

[30] A. Hans, D. Schneegaß, A. M. Schäfer, and S. Udluft, "Safe exploration for reinforcement learning," in *Proc. ESANN*, Apr. 2008, pp. 143–148.

[31] M. Pecka, K. Zimmermann and T. Svoboda, "Safe exploration for reinforcement learning' in real unstructured environments," in *Proc. Comput. Vis. Winter Workshop*, Seggau, Austria, Feb. 2015, pp. 85-93

[32] F. J. G. Polo and F. F. Rebollo, "Safe reinforcement learning in high-risk tasks through policy improvement," in *Proc. IEEE Symp. Adapt. Dynamic Program. Reinforcement Learn. (ADPRL)*, Apr. 2011, pp. 76–83.

[33] T. M. Moldovan and P. Abbeel, "Safe exploration in Markov decision processes," in *Proc. 29th Int. Conf. Mach. Learn.*, 2012, p. 188.

[34] L. P. Kaelbling, M. L. Littman, and A. W. Moore, "Reinforcement learning: A survey," *J. Artif. Intell. Res.*, vol. 4, no. 1, pp. 237–285, Jan. 1996.

[35] C. Szepesvári, "Algorithms for reinforcement learning," *Synth. lectures Artif. Intell. Mach. learn.*, vol. 4, no. 1, pp. 1–103, 2010, doi:10.2200/S00268ED1V01Y201005AIM009

[36] P. Geibel, "Reinforcement learning with bounded risk," in *Proc. ICML*, 2001, pp. 162–169.

[37] T. Fraichard and H. Asama, "Inevitable collision states—A step towards safer robots?" *Adv. Robot.*, vol. 18, no. 10, pp. 1001–1024, 2004.

[38] C. J. C. H. Watkins and P. Dayan, "Q-learning," *Mach. Learn.*, vol. 8, nos. 3–4, pp. 279–292, 1992.

[39] R. E. Moore, *Interval Analysis*, vol. 4. Englewood Cliffs, NJ, USA: Prentice-Hall, 1966.

[40] O. Khatib, "Real-time obstacle avoidance for manipulators and mobile robots," *Int. J. Robot. Res.*, vol. 5, no. 1, pp. 90–98, 1986.

[41] K. Ok, S. Ansari, B. Gallagher, W. Sica, F. Dellaert, and M. Stilman, "Path planning with uncertainty: Voronoi uncertainty fields," in *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, May 2013, pp. 4596–4601.

[42] S. G. Loizou, H. G. Tanner, V. Kumar, and K. J. Kyriakopoulos, "Closed loop motion planning and control for mobile robots in uncertain environments," in *Proc. 42nd IEEE Conf. Decision Control*, vol. 3, Dec. 2003, pp. 2926–2931.

[43] V. J. Lumelsky and A. A. Stepanov, "Path-planning strategies for a point mobile automaton moving amidst unknown obstacles of arbitrary shape," *Algorithmica*, vol. 2, nos. 1–4, pp. 403–430, 1987.

[44] K. P. Tee, S. S. Ge, and E. H. Tay, "Barrier Lyapunov functions for the control of output-constrained nonlinear systems," *Automatica*, vol. 45, no. 4, pp. 918–927, Apr. 2009.

[45] P. Wieland and F. Allgöwer, "Constructive safety using control barrier functions," *IFAC Proc. Volumes*, vol. 40, no. 12, pp. 462–467, 2007.

[46] M. Z. Romdlony and B. Jayawardhana, "Uniting control Lyapunov and control barrier functions," in *Proc. 53rd IEEE Conf. Decision Control*, Dec. 2014, pp. 2293–2298.

[47] T. M. Lam, H. W. Boschloo, M. Mulder, and M. M. Van Paassen, "Artificial force field for haptic feedback in UAV teleoperation," *IEEE Trans. Syst., Man, Cybern. A, Syst. Humans*, vol. 39, no. 6, pp. 1316–1330, Nov. 2009.

Tommaso Mannucci received the B.Sc. degree in aerospace engineering with the University of Pisa, Pisa, Italy, in 2009, and the M.Sc. degree in aerospace engineering and flight dynamics University of Pisa, Pisa, Italy, in 2012, with a thesis on numerical optimization for attitude recovery. He is currently pursuing the Ph.D. degree with the Control and Simulation Division, Aerospace Faculty, Delft University of Technology, Delft, The Netherlands.

His current research interests include adaptive control, machine learning, and reinforcement learning for UAVs.

Erik-Jan van Kampen received the B.Sc. degree in aerospace engineering, the M.Sc. degree in control and simulation, and Ph.D. degree in aerospace engineering from the Delft University of Technology, Delft, the Netherlands, in 2004, 2006, and 2010, respectively.

He is currently an Assistant Professor with the Control and Simulation Division, Delft University of Technology. His current research interests include intelligent flight control, adaptive control, and interval optimization.

Cornelis de Visser received the M.Sc. degree from the Delft University of Technology, Delft, The Netherlands, in 2007, and the Ph.D. degree from the Faculty of Aerospace Engineering, Delft University of Technology, in 2011.

He is currently an Assistant Professor with the Control and Simulation Section, Delft University of Technology. His current research interests include aircraft system identification, flight envelope prediction, fault tolerant control, and multivariate spline theory.

Qiping Chu received the Ph.D. degree from the Faculty of Aerospace Engineering, Delft University of Technology, Delft, The Netherlands, in 1987.

He is currently an Associate Professor with the Faculty of Aerospace Engineering, Delft University of Technology, and the Head of Aerospace Guidance, Navigation and Control Cluster within the Section of Control and Simulation. He has co-authored over 200 journal and conference papers ranged from adaptive control, nonlinear control, robust control and intelligent control to nonlinear state estimation, system identification, and nonlinear optimization for aerospace vehicles.