A novel approach with safety metrics for real-time exploration of uncertain environments

Mannucci, T; van Kampen, EJ; de Visser, CC; Chu, QP

DOI
10.2514/6.2016-0637

Publication date
2016

Document Version
Accepted author manuscript

Published in
Proceedings of the AIAA guidance, navigation, and control conference

A novel approach with safety metrics for real-time exploration of uncertain environments

T. Mannucci*, E. van Kampen †, C. C. de Visser ‡and Q. P. Chu§
TU Delft, Delft, Zuid Holland, the Netherlands

Various research has been done on the application of Reinforcement Learning for adaptive controllers for aerospace, due to its core simplicity of design and its model-free capabilities resulting in a great flexibility of application. During real-life exploration of the environment, such a controller will employ various algorithms to accelerate the collection of significant data and therefore the convergence of the value function. If the environment presents any form of danger for the agent, these algorithms need to cope with the additional requirement of avoiding actions leading to such dangers, even when a definite model of the agent in the environment is not available. In this paper, computing a safety-weighted graph based on a tiling of the state space, and with the introduction of two different metrics for action selection is shown as a promising method for avoiding dangers during exploration. As proof of concept, the method is applied on two simulated tasks: a high-level navigation task for an autonomous UAV, and a classical, low-level task of controlling the elevator deflection of an aircraft.

Nomenclature

RL	Reinforcement Learning
MDP	Markov Decision Process
$\mathcal{S}$	state space
$\mathcal{A}$	action set
$\mathcal{D}$	dynamic law
$\mathcal{F}$	fatal function
$\mathcal{W}$	warning function
$\mathcal{G}$	dynamical graph
OM	Operative Metric
PM	Proximity Metric

I. Introduction

Reinforcement learning[1] (RL) is a popular control scheme in different branches of control. The core concept of RL stems from animal adaptation to the environment. Depending on the *situation* provided by the environment, the animal must adopt a *behavior* that allows to perform a task with varying degree of success. If the task is correctly performed (e.g. the animal manages to find and eat food) then a positive *stimulus* further roots the behavior for the animal. RL replicates this approach: an agent evaluates actions under different states of the environment in order to collect reward, in a process called *exploration*. The agent will then use the reward information to synthesize a policy with the goal of performing best at a certain task: this is the *exploitation* phase. Inadequate or insufficient exploration will result in unsatisfactory exploitation,

*PhD student, Aerospace Faculty, Control & Operations Department
†Assistant professor, Aerospace Faculty, Control & Operations Department
‡Assistant professor, Aerospace Faculty, Control & Operations Department
§Associate professor, Aerospace Faculty, Control & Operations Department

but unnecessary exploration should be avoided. During learning, a conflict arises between exploring more, training the agent further, or start exploiting the current knowledge to increase performance in the task.

In this paper another conflict will be considered, the one between performance and *safety*. In most RL applications the notion of safety is usually overlooked: the behavior of the agent is driven by the more pressing concern of exploration-exploitation, with safety being either a non-issue or embedded inside the concept of reward[2].[3] Examples where these are not realistic approaches are abundant in real-life applications and even more so in aerospace applications. For example, an airborne RL controller is constrained by the aircraft's flight envelope. Piloting a wheeled robot (e.g a rover) would require careful avoidance of high risk environments such as cliffs. In all the previous examples, if the RL controller was left in complete autonomy in selecting actions, at most driven by performance concerns, exploration would be vulnerable to incurring in dangerous and possibly fatal situations. On the other hand, an overly conservative, "fearful" exploration could result in an equally unacceptable controller that doesn't learn enough to perform its task even when a reasonably safe policy is available.

The tradeoff between performance and safety was addressed in previous work. Hans et al.[4] (2008) propose an algorithm for plant control that avoids fatal transitions; however the algorithm relies on an a-priori known safety function (acting as a go/no-go decision maker over possible actions) and a fixed backup policy valid in all workspace. Garcia and Fernandez[5] (2011) have a similar approach; introducing a variable amount of perturbation in given safe but inefficient baseline controller, discovery of new trajectory for task completion is possible (taking however a certain degree of risk). The two share the need of a guaranteed safe controller or backup policy in order to prevent catastrophic exploration when facing critical decisions. Moldovan and Abbeel[6] (2012) define safety in terms of ergodicity of the exploration, and introduce an algorithm that still relies on believes of the system but not on a predefined baseline policy or safe controller. Safety of the exploration is again guaranteed within a certain degree of reliability. Gillula et al.[7] (2010) and Gillula and Tomlin[8] (2011), while not dedicated to RL exploration, show very promising applications of reachability analysis to the problem of planning safe control.

It was showed in previous work[9] how the problem of safety could be addressed by looking, in near time, to possible *backups*. When following a backup, the controller would be able, in near-time, to bring the system in a close neighborhood of a state that was previously visited. The controller would heuristically search for backups at each time-steps, and refrain from taking actions for which no backups could be found. With respect to the previous methods, this approach does not resort to an a-priori known safe policy, nor does rely on any hypothesis on the system at hand. Instead, the benefit of such a scheme is to automatically induce a cautious behavior in the agent. Extreme commands, for which the agent drifts away from the already known situations are discarded in favor of more careful ones.

In this paper, this approach is further investigated under a different methodology. First, the state space is turned into a discrete tiling. Then a weighted graph is generated from the uncertain, possibly non linear dynamics, with the vertices representing elements of the tiling, and the weights representing a current estimate of safety for the states of the vertex. The computation of a backup as an heuristic search is then replaced by minimization of a metric over the available action set, leading to the highest level of safety in near-time. This new methodology results in lighter computational load with ad-hoc scalable complexity.

The rest of the paper is structured as follows. In section II, the problem of safe exploration, and the hypothesis upon which this work relies will be discussed. In section III, the mathematical framework upon which the method relies will be introduced in the form of a graph generating procedure, and of two safety metrics. In section IV the algorithm itself will be thoroughly discussed. In section V two simulated applications of the method will be presented: a quadrotor navigation task, and an elevator control task. Finally in section VI the conclusions will be drawn.

The following notation will be adopted. Bold characters and brackets will be used to indicate vector quantities. Square brackets will indicate intervals, while curly brackets will indicate sets. Infimum and supremum of interval i will be indicated respectively as $\underline{i}$ and $\overline{i}$. Symbol $\cdot$ will represent scalar product, $\times$ will represent set combination, and $*$ will represent elementwise product.

II. Fundamentals of Reinforcement Learning in dangerous environments

A. Reinforcement learning

This section will present a classic framework of RL, known as a Markov Decision Process (MDP). It can be identified by a tuple of five elements: state, action, transition, reward and discount. Let $\mathcal{S}$ be the set of all

possible states that the system can assume. In case of tasks where multiple states are considered, e.g. the agent internal state and the environment external state, $\mathcal{S}$ would consist of the combination of the two. In most real-life tasks $\boldsymbol{s} \in \mathcal{S}$ is a state vector whose components represent various physical quantities. In the present work , $\mathcal{S}$ will be an *hybrid* space with elements $\boldsymbol{s}$ in the form

$$\boldsymbol{s} = (x_1,\, x_2,\, \cdots,\, x_m,\, z_1,\, \cdots,\, z_n). \tag{1}$$

with the generic continuous coordinate $x_i \in [\underline{x}_i,\ \overline{x}_i] \subseteq \mathbb{R}$ and with the generic discrete coordinate $z_j \in \{z_{j1}, \cdots, z_{jk}\} \subset \mathbb{R}$. This formulation can represent physical systems with logical or discrete attributes. In the event that $\mathcal{S}$ is purely continuous, it will be informally referred to as *natural*.

Let $\mathcal{A}(\boldsymbol{s}, t)$ be the set of available actions to the agent and controller. For each couple of state and action, transitions between states are governed by a *dynamic law*. The formulation for such a law drastically changes with the model. For purely discrete systems, $\mathcal{D} : \mathcal{S} \times \mathcal{A} \to \mathcal{S}$. For purely continuous systems, $\mathcal{D} : \mathcal{S} \times \mathcal{A} \to \mathbb{R}^{(m)}$. For hybrid systems such as hybrid automatas, the formulation can get more complex.[10] In this paper it will be assumed that $\mathcal{D}$ can be written as

$$\mathcal{D} : \begin{cases} \dot{x}_1(t) = \mathcal{D}_1\,(x_1(t),\, \cdots,\, x_m(t),\, z_1(t),\, \cdots,\, z_n(t),\, a) \\ \qquad\qquad\qquad\qquad \vdots \\ \dot{x}_m(t) = \mathcal{D}_m\,(x_1(t),\, \cdots,\, x_m(t),\, z_1(t),\, \cdots,\, z_n(t),\, a) \\ z_1(t+dt) = \mathcal{D}_{m+1}\,(x_1(t),\, \cdots,\, x_m(t),\, z_1(t),\, \cdots,\, z_n(t),\, a) \\ \qquad\qquad\qquad\qquad \vdots \\ z_n(t+dt) = \mathcal{D}_{m+n}\,(x_1(t),\, \cdots,\, x_m(t),\, z_1(t),\, \cdots,\, z_n(t),\, a). \end{cases} \tag{2}$$

Function $\mathcal{D}$ of Eq. (2) depends only on current state and action: this is the *Markov property*. Additionally, $\mathcal{D}$ is assumed not to be explicitly dependent on time. The fourth element of the tuple is a *reward function* $\mathcal{R} : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$, which can be stochastic. After each action, the agent receives the reward r yielded by the function, which represents an immediate, short-term benefit. The goal of the agent is to maximize the long-term benefit in the form of the cumulative expected reward:

$$J(\boldsymbol{s}, \pi) = E\left(\sum_{k=0}^{\infty} \gamma^k r_k\right) \tag{3}$$

where $\gamma < 1$ is a discount factor. Obtaining the optimal policy

$$\pi^*(\boldsymbol{s}) : \pi^* = \operatorname{argmax} J(\boldsymbol{s}', \pi),\ \ \forall \boldsymbol{s}' \in \mathcal{S} \tag{4}$$

is in general a difficult task. $\mathcal{D}$ can be unknown or only approximately known. Reward function $\mathcal{R}$ can be hidden or stochastic; it can also be poorly informative, e.g. only yielding a reward at the end of the task, and none during the task itself. Also, the state space $\mathcal{S}$ can be vast even for relatively simple problems: most algorithms have proof of finding π^* only when $\mathcal{S}$ is finite. The more difficult the task, the more exploration will be needed to obtain a satisfactory policy.

B. Safe exploration

If the task presents any form of danger, a second conflict must be resolved between exploration and safety. In order to do so, it is useful to extend the previous framework with additional elements. A fatal function $\mathcal{F} : \mathcal{S} \to \{0, 1\}$ indicates whether a state is either *safe* or *fatal*. A fatal state can be seen as a state for which the agent encounters an unacceptable condition; for example if the agent is harmed, e.g. a crash or a failure, or if it cannot proceed further in its task, e.g. running out of fuel or battery, or even if the agent damages its surroundings, e.g. hurting a human user. This purposely broad definition includes all those events that, while not directly part of the agent's task, must be avoided regardless of future cumulative reward, or else the task is considered to fail. A warning of danger denominated *risk perception*[9] is assumed in the form of an unknown function $\mathcal{W} : \mathcal{S} \to \{0, 1\}$ that indicates if danger is perceived within a known neighborhood of the current state $\boldsymbol{s}$. This *warning* function will be used by the agent to individuate fatal states and avoid them. This definition is sufficiently generic to include information derived from sensors, discrete warnings and expert input alike.

C. Bounding dynamic law

In the event of $\mathcal{D}$ being unknown[11] or approximated,[12] various algorithms exist to to discover an appropriate policy. Theoretically speaking, the agent could perform trial-and-error investigations, accumulating reward and at the same time updating its policy in the total absence of a model, or by observing transitions to improve its approximation of $\mathcal{D}$. The first class of *model-free* methods is not ideal for handling safe exploration, since it is inherently based on reward, to which fatal states are not associated. The second class of *model-based* methods is more apt for such a task, since it allows to predict future states of the system. However, unless a perfect model is available, the consequent error in the prediction means that prevention of the fatal states depends on the precision of the approximation. Consider those cases where the uncertainty can be bounded. Then, predictions on future states of the system can be handled by such tools as interval analysis[13] to yield a *bounding law*

$$\hat{\mathcal{D}} : \begin{cases} \dot{x}_1(t) \in \hat{\mathcal{D}}_1 \left(x_1(t), \cdots, x_m(t), z_1(t), \cdots, z_n(t), a\right) \\ \qquad\qquad \vdots \\ \dot{x}_m(t) \in \hat{\mathcal{D}}_m \left(x_1(t), \cdots, x_m(t), z_1(t), \cdots, z_n(t), a\right) \\ z_1(t+dt) \in \hat{\mathcal{D}}_{m+1} \left(x_1(t), \cdots, x_m(t), z_1(t), \cdots, z_n(t), a\right) \\ \qquad\qquad \vdots \\ z_n(t+dt) \in \hat{\mathcal{D}}_{m+n} \left(x_1(t), \cdots, x_m(t), z_1(t), \cdots, z_n(t), a\right). \end{cases} \tag{5}$$

The difference between the actual $\mathcal{D}$ and $\hat{\mathcal{D}}$ is in that while the first predicts $\boldsymbol{s}$ exactly, the second predicts a set $\{\boldsymbol{s}\}$ such that $\boldsymbol{s} \in \{\boldsymbol{s}\}$. An immediate consequence of such a modelization is that, while it allows to take into account the uncertainty inherent to the system, predicted trajectories tend to bloat in time as uncertainties accumulate.

D. Lead-to-fatal states

Freichard and Asama[14] introduced the notion of *inevitable collision states* for robots in obstacle avoidance tasks. These are all those combinations of speed and positions for which a collision with an obstacle is inevitable. Applying this same concept to the fatal states, viewing them as "obstacles" inside $\mathcal{S}$, will yield the lead-to-fatal (LTF) states. A LTF state is a condition of the agent from which all possible future trajectories intersect the fatal set $\boldsymbol{s} : \mathcal{F}(\boldsymbol{s}) = 1$. If an agent assumes a LTF state, it will have a fatal occurrence sometimes in the future. Avoiding LTF states is just as important as avoiding fatal states, but while the latter can be perceived by means of risk-perception, the former cannot. This problem is aggravated by the bloating of predicted trajectories generated by the uncertain law $\hat{\mathcal{D}}$. In previous work,[9] a possible approach to solving this problem was found in constraining the evolution of the system to a neighborhood of previously visited and reliable conditions. This approach will be followed in this work as well by selecting actions that optimize a safety metric.

E. Assumptions

In the remainder of this paper the following assumptions will be made:

1. $\mathcal{S}$ will be considered to be as indicated by Eq. (1);
2. $\mathcal{A}$ will be considered to be state and time independent;
3. $\hat{\mathcal{D}}$ is a time-independent bounding law of $\mathcal{D}$;
4. $\mathcal{F}$ and $\mathcal{W}$ will be considered to be time independent but otherwise unknown.

III. Tiling approach and metrics

This section is divided in two parts. In the first part, it will show a method to obtain a directed graph $\mathcal{G}$ whose vertices will represent the states of the system, and whose edges will represent transitions between

states through actions. The procedure to generate the graph follows three steps. First, the state space is partitioned into tilings, each representing a vertex. Then, the action set $\mathcal{A}$ is converted into a representative subset to reduce the complexity of the graph. Third, the bounding law $\hat{\mathcal{D}}$ is applied to connect the state vertices through action edges. In the second part of the section, two metrics will be discussed. Each metric assigns a value to each action of the agent depending on the predicted trajectory, given the current knowledge of the environment and the previous history of the exploration. This value will indicate which action is the safest.

A. Tiling and graph generation

1. *Tiling*

As a first step, the state space $\mathcal{S}$ is partitioned into a *tiling* by mean of tile coding.[15] Excluding tile borders, each element of $\mathcal{S}$ belongs to one tile. Different tiles do not need to be identical or to follow a definite pattern: in various applications, the size and shape of the tiles vary locally and even adaptively.[16] However, a tiling with identical tiles is considered in this work for reasons that will become clearer in the following. Such a tiling can be seen as the result of evenly partitioning the continuous coordinates of $\mathcal{S}$ into intervals of fixed width Δ_i:

$$[\underline{x}_i,\ \ \overline{x}_i] = [\underline{x}_i,\ \ \underline{x}_i + \Delta_i] \ \cup\ [\underline{x}_i + \Delta_i,\ \ \underline{x}_i + 2\Delta_i] \ \cup \cdots \cup\ [\overline{x}_i - \Delta_i,\ \ \overline{x}_i] \tag{6}$$

so that each tile represents a unique combination of continuous intervals and discrete components:

$$[\underline{x}_1 + (\tau_1 - 1)\cdot\Delta_1,\ \ \underline{x}_1 + \tau_1\cdot\Delta_1] \times \cdots \times [\underline{x}_m + (\tau_m - 1)\cdot\Delta_m,\ \ \underline{x}_m + \tau_m\cdot\Delta_m] \times z_{1\tau_{m+1}} \times \cdots \times z_{n\tau_{m+n}} \tag{7}$$

where vector $\boldsymbol{\tau} = (\tau_1, \cdots, \tau_{m+n})$ is the *index* of the tile, indicating its position inside the whole tiling. Figure 1 illustrates an example of such a tiling. Each tile will constitute a vertex in the final graph.

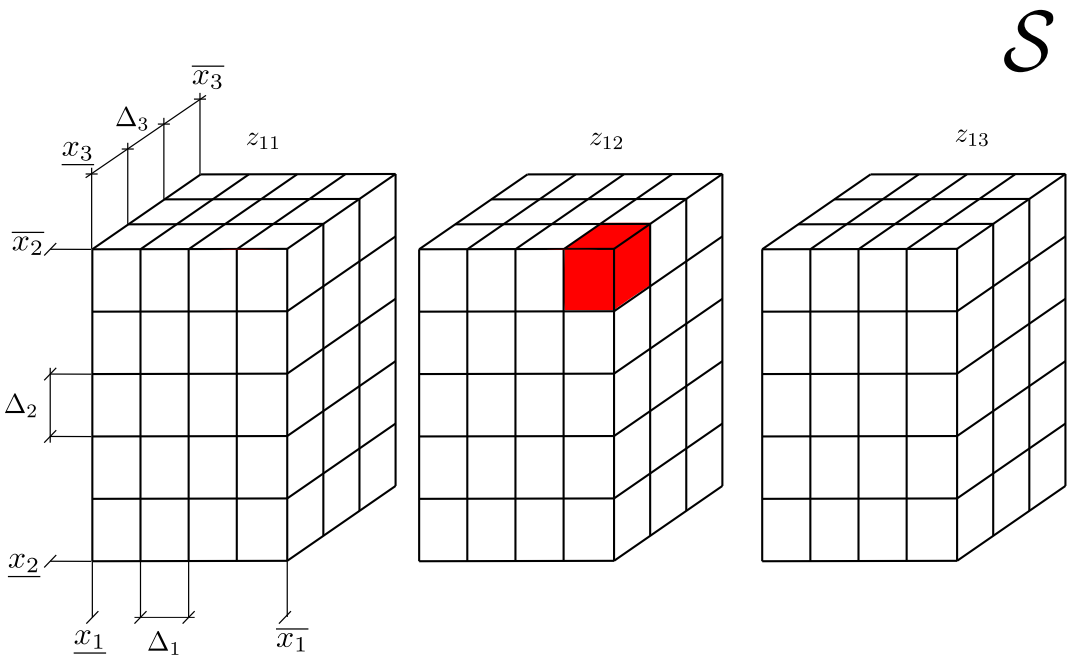


Figure 1: A simple example of an even tiling of an hybrid system with 3 continuous components x_1, x_2 and x_3, and with one discrete component z_1. The tile with index $\boldsymbol{\tau} = (4, 5, 1, 2)$ is shaded in red. All tiles have the same size due to each component having been evenly divided.

2. *Actions*

Actions of the agent determine transitions between states of the system, and are thus represented in the graph as edges between vertices. In theory, the agent can perform any of the actions in set $\mathcal{A}$. However, the

more actions available to the agent, the more the number of outbound edges per vertex and consequently the more complex the graph. In the limit, if $\mathcal{A}$ is not finite, an infinite number of edges should be generated. Therefore, as a second step, a representative subset $\mathcal{A}_{sub}$ is extracted from the action set $\mathcal{A}$. Limiting the choice of $\mathcal{A}_{sub}$ to a reasonable amount is key to speeding up the graph generation.

3. Graph generation

Having the vertices given by the tiling, and the edges given by subset $\mathcal{A}_{sub}$, the bounding law $\hat{\mathcal{D}}$ of Eq. (5) is invoked to generate the graph. One convenient form for $\hat{\mathcal{D}}$ is the interval form

$$\hat{\mathcal{D}}_i = [\underline{\dot{x}}_i,\ \overline{\dot{x}}_i],\ i = 1,\ \ldots,\ m. \tag{8}$$

Interval notation for $\hat{\mathcal{D}}$ comes natural when considering systems whose uncertainty derives from parameters which are intervals. It is always possible to switch to this notation by considering the highest and the lowest value among the set of the possible outputs $\hat{\mathcal{D}}_i$. It will be thus assumed that such a formulation is available. As a further step, the dynamics will be discretized in time. The time-step Δt should be chosen with the same order of magnitude of the fastest dynamics. However, the shorter the time-step, the more the tiling must be refined: the reason for this will be explained later in this section. After this last iteration, the discrete dynamics will be in the following form:

$$\hat{\mathcal{D}} : \begin{cases} x_1(t+\Delta t) \in \hat{\mathcal{D}}_1\ (x_1(t),\ \cdots,\ x_m(t),\ z_1(t),\ \cdots,\ z_n(t),\ a(t)) \\ \qquad\qquad \vdots \\ x_m(t+\Delta t) \in \hat{\mathcal{D}}_m\ (x_1(t),\ \cdots,\ x_m(t),\ z_1(t),\ \cdots,\ z_n(t),\ a(t)) \\ z_1(t+\Delta t) \in \hat{\mathcal{D}}_{m+1}\ (x_1(t),\ \cdots,\ x_m(t),\ z_1(t),\ \cdots,\ z_n(t),\ a(t)) \\ \qquad\qquad \vdots \\ z_n(t+\Delta t) \in \hat{\mathcal{D}}_{m+n}\ (x_1(t),\ \cdots,\ x_m(t),\ z_1(t),\ \cdots,\ z_n(t),\ a(t)) \end{cases} \tag{9}$$

The procedure for generating the graph $\mathcal{G}$ is now straightforward. Each vertex will represent a tile. Given a tile and an action $a \in \mathcal{A}_{sub}$, Eq. (9) is executed for all states in the tile to yield the set of possible next states. A directed edge, labeled with the current action, is drawn from the "starting" tile/vertex to all tiles/vertices with a non-empty intersection with this set. Note that the result is technically a multi-graph, since each edge is possibly connected to multiple vertices. If any state of the generated set is not in $\mathcal{S}$, then the action is forfeited. No edge labeled with this action is generated. The need for a trade-off in terms of time-step and coarsity of the tiling can be now explained. If in a time-step the system transitions from a state belonging to a tile to a different state in a different tile, this transition is shared, in the graph formulation, by all states belonging to the starting tile. If the tiles are too large when compared to the time-step, sharing this transition will result in an artificially accelerated representation of the dynamics, hence the need for a trade-off when selecting the time-step value.

The interval formulation of the dynamics and the fixed-grid tiling reduce the computational burden of the graph generation. For example, if state x_i increases of an amount comprised between $\underline{\Delta x}_i$ and $\overline{\Delta x}_i$, in terms of tiling representation that means increasing component τ_i of the index of an amount comprised between $\underline{\Delta x}_i/\Delta_i$, rounded down, and $\overline{\Delta x}_i/\Delta_i$, rounded up. This computational advantages make for a quick and robust graph generation. A natural drawback of the using partitions with fixed widths is that the coarsity of the tiling cannot be increased or decreased locally. While this is inefficient, it should be noted that, in the absence of any form of tiling-refinement method,[17] selecting a non-uniform tiling is a difficult problem that requires prior knowledge of the case in exam.

B. Metrics

In this section we will introduce two metrics: an *operative metric* (OM) that will embed information deriving from the warning function, and a *proximity metric* (PM) that will account for the degree of exploration of the system in near-time. Both metrics can be applied to either one vertex or to a collection C of vertexes, and the output will depend on the current state of the exploration.

1. *Operative metric*

$\mathcal{F}$ and $\mathcal{W}$ are functions defined over the original $\mathcal{S}$, and hidden to the agent. The goal of the OM is to embed this information into an approximation of the function $\mathcal{F}$ that can be readily relied upon by the agent at each time step. Define four real valued quantities $q_{exp} > q_{safe} \gg q_{unc} \gg q_{fat}$. At the moment of graph generation, all vertices are initialized with a value equal to q_{unc} representing the notion that $\mathcal{S}$ is unknown at the start. When $\mathcal{W}(\boldsymbol{s})$ is invoked, if no risk is perceived, all tiles that entirely fall within the perception range are labeled as safe. The values of vertices corresponding to the safe tiles is replaced by q_{safe}. Tiles whose elements are only partially in range or not in range are unaltered. Finally, the value of current vertex/tile will be updated to q_{exp}. Conversely, in the event that risk is perceived, at least one of the tiles currently in range contains a fatal state. Therefore, all tiles that fall even partially in range of the risk perception, and that are still unexplored (i.e. whose vertices have value q_{unc}) are considered potentially fatal: their value is updated to q_{fat}. For a collection of vertices C the value will be the average q of all vertices v in the collection. Value replacement is applied at every time-step of the exploration to increasingly improve the agent's approximation of $\mathcal{F}$. It should be noted that individuating which tiles fall in range of the perception is simplified by the use of an even tiling. An example of the application of the OM is shown in figure 2.

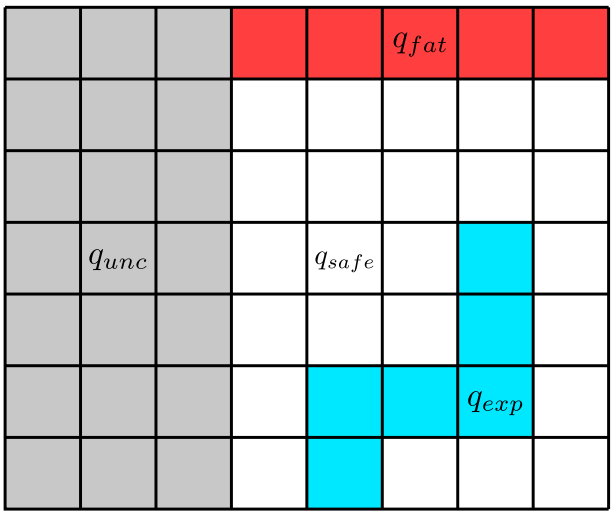


Figure 2: An example of the operative metric. Explored tiles (blue) are assigned the highest value q_{exp}. Safe tiles (blank) are assigned a reduced value q_{safe} which is significantly higher than the value q_{unc} assigned to unexplored tiles (grey). Finally, those states that have been perceived as possibly fatal (red) are assigned the lowest value q_{fat}.

2. *Proximity metric*

As the name suggests, the PM evaluates vertices with respect to their closeness to previously visited states. In order to account for closeness, a definition of *distance* between two vertices v and v' is introduced as

$$\text{dist}(v,\ v') = \left\| \boldsymbol{\rho} * (\boldsymbol{\tau} - \ \boldsymbol{\tau}') \right\|_2 \tag{10}$$

where τ is the index of v, τ' is the index of v', and $\boldsymbol{\rho} \in \mathbb{R}^{m+n}$ is a vector of positive weights. Essentially, the distance between tiles is computed as the rescaled norm of the difference in position inside the tiling. When considering a system with only continuous components, this distance is the tiling equivalent of computing the Eulerian distance between two states in a rescaled state space. As for the discrete components of the state, the assumption is made here that the discrete values can be ordered in such a way that the difference in indexing is still indicative of a progressively changing condition. The term $\boldsymbol{\rho}$ acts as a rescaling vector for state space $\mathcal{S}$: depending on the weights assigned, the same difference in index of two components will have a different contribution to the metric. This can be used to include previous knowledge into the definition of distance. For example, more relevant components of the state could be assigned a higher weight than less influential or more easily controllable components.

Now that a distance is introduced, the metric can be properly discussed. At each time-step, the controller observes its current state and adds it to a list of previously visited states $\mathcal{S}_{\text{list}}$. Given a vertex v and a list $\mathcal{S}_{\text{list}}$, the following metric can be applied:

$$\text{prox}(v,\ \mathcal{S}_{\text{list}}) = -\min_{\boldsymbol{s}' \in \mathcal{S}_{\text{list}}} \text{dist}(v,\ v' \,|\boldsymbol{s}' \in v') \tag{11}$$

i.e. the proximity of a tile is its distance to the nearest tile containing an explored state, changed in sign. The higher the proximity, the lesser the current state differs from a state already visited and thus known; conversely, the lower the proximity, the more unknown the state. The following extension is applied when considering a collection of states C. First, the center c of the collection is found. Then the proximity of C is equal to the proximity of c plus an additional uncertainty term:

$$\text{prox}(C,\ \mathcal{S}_{\text{list}}) = \text{prox}(c,\ \mathcal{S}_{\text{list}}) - \eta \max_{v' \in C} \text{dist}(c, v') \tag{12}$$

with a positive weighting term $\eta < 1$. This additional term is proportional to the distance of c from the furthest tile of the collection. Therefore, applying this metric not only accounts for the mean distance between a tile and a collection, but also for the dispersion in the collection. A. Figure 3 shows an example of the application of the metric. The state space has two continuous components x_1 and x_2. The slight grey square represent the collection C, with its center c. The blue tiles represent those tiles containing a visited state $\boldsymbol{s} \in \mathcal{S}_{\text{list}}$. With a weight vector $\boldsymbol{\rho} = (2,\ 1)$, tile v'_s is the one with the highest proximity of $-4\sqrt{2}$, higher than that of v''_s which is equal to -8. Therefore v'_s is the "nearest" tile under this metric even though v''_s is nearer inside the unweighted tiling. Finally, a term proportional to the distance between c and v_f, the furthest tile of the collection, must be added to compute the proximity of the whole collection.

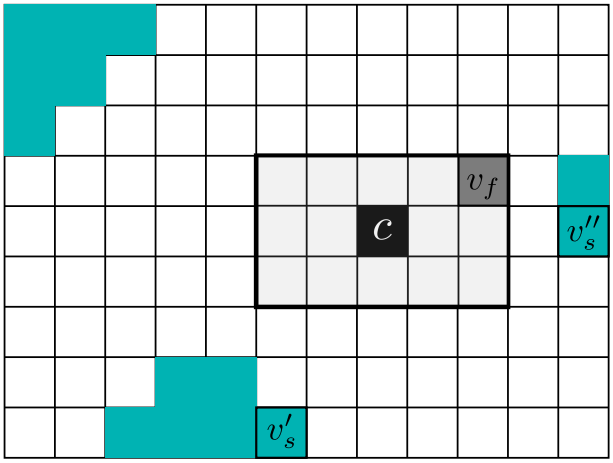


Figure 3: An example of proximity computation for collection C. The black tile c is the center of the collection. The blue tiles are tiles containing a visited state. Among these, with the assigned weight vector, v'_s is the nearest tile to c. A term proportional to the distance between c and v_f must then be added to compute the proximity.

IV. Algorithm description

This section will illustrate the algorithm for safe exploration in detail. Initially, $t = t_0$, $\boldsymbol{s} = \boldsymbol{s}_0$. It will be assumed that at the start of the exploration no risk is perceived, i.e. $\mathcal{W}(\boldsymbol{s}_0) = 0$. The goal of the agent is to select an action among the available set $\mathcal{A}_{sub}$ which will keep exploration safe. Actions are considered in the form of *commands* $\boldsymbol{a} = \{a(t),\ a(t + \Delta t),\ \cdots,\ a(t + k \cdot \Delta t)\}$. The graph $\mathcal{G}$ can be invoked to predict the final state of the system after the application of a command. This final state can be evaluated with a safety metric: the command that optimizes the metric is the safest. This approach can then be seen as a variant of Model Predictive Control.[18] However, given the formulation of the metrics and the uncertainties in the available predictions, it is not advisable to look for a solution to the optimization problem in close form. Instead, the optimal command will be selected among a restricted selection of candidates.

Although any set of candidate commands can be evaluated under the proposed metrics, the following restrictions were imposed in this work. A first restriction comes from noticing that there is a limit on how many steps ahead can be efficiently predicted by $\mathcal{G}$. Each edge of the graph can connect the starting vertex to more than one arrival vertex, due to the the inherent uncertainties in $\hat{\mathcal{D}}$. As a result, predictions of arrival tiles tend to bloat, and are less and less useful with the increase of the time steps. So, candidate commands will have a duration in time equal or shorter than a predefined number of steps k_{max}. This limits the set to a finite number of candidates. Depending on the application in exam and the duration of a time-step, a lower limit k_{min} on the length of the command might also be imposed. This is due to the fact that some dynamics might be slower than others. Then a minimum amount of iterations are needed to observe the effect of the command in said dynamics. A further selection will be made by considering candidates in the form:

$$\boldsymbol{a} : a(t) = a(t + \Delta t) = \cdots = a(t + k \cdot \Delta t) = \bar{a} \tag{13}$$

i.e. constant commands. The reason for this choice is the following. When considering commands lasting considerably in time, the optimal command could be expected to present significant variations in the actions involved. However, when considering commands that are severely limited in time (such as those considered in this paper), it is more meaningful to consider constant commands that truly represent the effect of the atomic actions, rather than commands with mixed actions whose effects might be conflicting. This selection reduces the number of metric evaluations per time-step to a fixed amount, i.e. the cardinality of $\mathcal{A}_{sub}$ times the number of allowed time-steps $k_{max} - k_{min}$.

Figure 4 summarizes the algorithm. The composing element of the algorithm are the graph $\mathcal{G}$, a predefined set of action $\mathcal{A}_{sub}$ generating a set of commands $\{\boldsymbol{a}\}$, a warning function $\mathcal{W}(\boldsymbol{s})$ and a safety metric. The system starts in state $\boldsymbol{s}_0$, which in the graph $\mathcal{G}$ corresponds to current vertex v_0. $\mathcal{G}$ can now predict the trajectory of the system under command $\boldsymbol{a} = \{a(t), a(t+\Delta t), \cdots, a(t+k\cdot\Delta t)\}$. First, follow the outbound edge of v_0 corresponding to action $a(t)$ to individuate the one-step ahead collection of vertices. From these, follow the outbound edges corresponding to action $a(t + \Delta t)$, to individuate the two-step ahead collection. By proceeding iteratively, individuate the final collection of states $C(\boldsymbol{a})$. The collection is evaluated by the metric to give the value of the command. After repeating this process for all candidate commands, the optimal command $\boldsymbol{a}^*$ is selected, and corresponding action $a^*(t)$ is performed in the system. The new state $\boldsymbol{s}_1$ is observed. Finally, the metric history of exploration is updated with the previous state $\boldsymbol{s}_0$, and the approximation of $\mathcal{F}$ is updated with the current warning signal w. The process then repeats.

When individuating collections C, it can happen at any iteration that the outbound edge indicated by the current command is not present. This is because some state and action pairs can reach states outside of the state space $\mathcal{S}$, and therefore outside of the graph. This edges are excluded from $\mathcal{G}$ during graph generation. If a command indicates to perform an action for which no edge is available, then it is removed from the candidates for the current optimization. Also, it is important for the validity of the method to verify that arrival vertices C are as safe as possible. This is intrinsically included in the use of the operational metric due to the very low value of fatal states. If the chosen metric does not intrinsically include such a penalty, as with the proximity metric, a separate check must be performed, and if any command results in safety to be violated, it should be either discarded or heavily penalized.

V. Applications

This section will present two applications of the algorithm: a navigation task for a quadrotor and a control task for an aircraft with uncertain elevator dynamics. These particular tasks have been selected for two reasons. First, they represent two separated aspects of interest in current research on learning controllers, i.e. autonomous flight and in-flight fault management. Second, the tasks involve very different dynamics with different control challenges. For each task, a controller with the OM, a controller with the PM, and an exploratory controller selecting random actions are applied and compared.

A. Quadrotor navigation task

This simulated task consists in controlling a quadrotor inside a room, while avoiding hitting the walls. The quadrotor is equipped with sensors that allow to identify the walls at a given distance. The dynamics $\mathcal{D}$ of the quadrotor are schematically represented by the hybrid system of Eq. (14):

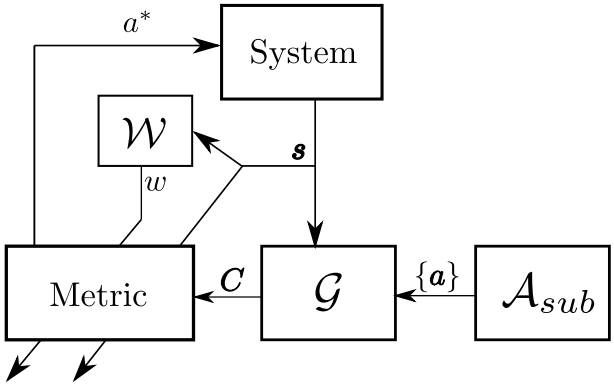


Figure 4: The algorithm for action selection. Given current state $\boldsymbol{s}$, the corresponding tile in graph $\mathcal{G}$ is individuated. Then, candidate commands $\boldsymbol{a}(t)$ are evaluated to yield vertices collections C. Each collection is evaluated under the current metric, and then optimal action a^* is performed. A new state is then generated from the system and observed. In addition, the metric is updated with the current status of exploration, provided by the state and by the warning function $\mathcal{W}$.

$$\dot{x} = V\ \cos(\psi);\ \dot{y} = V\ \sin(\psi);\ \dot{V} = \theta\,\dot{V}_c;\ \dot{\psi} = \begin{cases} +\dot{\psi}_c \text{ if } \texttt{clock} \\ -\dot{\psi}_c \text{ if } \texttt{c\_clock} \\ 0 \quad \text{else} \end{cases};\ \Delta\theta = \begin{cases} +1 \text{ if } \texttt{forw} \wedge \theta \neq +1 \\ -1 \text{ if } \texttt{back} \wedge \theta \neq -1 \\ 0 \quad \text{else} \end{cases} \tag{14}$$

where x and y indicate the position of the quadrotor, V and ψ respectively speed and heading, and θ the pitch configuration: positive, negative, or neutral pitch. The set of actions $\mathcal{A}$ comprises `forw` and `back` to increase and decrease the pitch; `clock` to steer clockwise and `c_clock` to steer counter-clockwise; and the neutral action `neut`. Eq. (14) was devised to account for the core dynamics of a generic quadrotor. To fit the model to a specific platform, the values of acceleration $\dot{V}_c$ and turning rate $\dot{\psi}_c$ can be specified to represent the actual performance. In this application, however, it will be assumed that the agent is unaware of the exact capabilities of the quadrotor, having at its disposal only an uncertain model $\hat{\mathcal{D}}$ obtained by replacing the true values $\dot{V}_c$ and $\dot{\psi}_c$ with their interval equivalent $\hat{\dot{V}}_c = [0.24,\ 0.6]\frac{\text{m}}{\text{s}}$ and $\hat{\dot{\psi}}_c = [\pi/4,\ \pi/3]$ s^{-1}. Substituting this intervals in Eq. (14) yields the uncertain dynamic law $\hat{\mathcal{D}}$ used by the controller.

It will now be shown how to generate the graph for the application of the algorithm. In this case, the state space $\mathcal{S}$ is not finite. A restriction of $\mathcal{S}$ to conform it to Eq. (1) will be then performed. States x and y are physically bounded by the dimension of the square room: $x, y \in [-5,\ 5]$ m. The angle ψ is bounded between $-\pi$ and π, and θ is already restricted in the formulation of the dynamics. Therefore, only the speed V needs to be artificially restricted. In the present work, $V \in [-1.2,\ 1.2]\ \frac{\text{m}}{\text{s}}$ is selected. This value is high enough to provide an efficient exploration, but not too challenging for the controller.

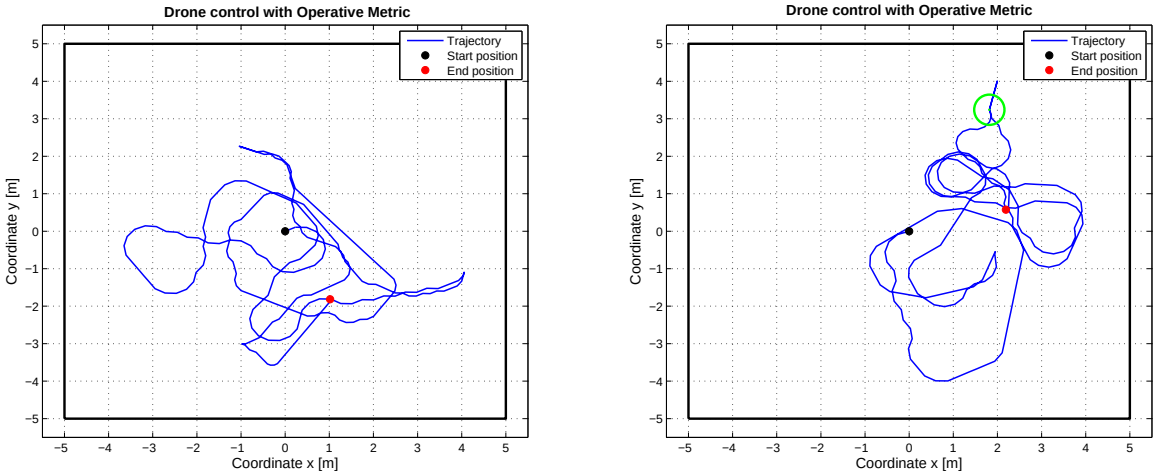
Then, all continuous coordinates of the restricted $\mathcal{S}$ are evenly divided into 20 intervals; discrete configuration θ is left unaltered. This results in a finite tiling of $4.8 \cdot 10^5$ tiles. The action set $\mathcal{A}$ is already atomic, so $\mathcal{A}_{sub} = \mathcal{A}$. The bounding law $\hat{\mathcal{D}}$ is obtained by replacing $\dot{V}_c$ with $\hat{\dot{V}}_c$ and $\dot{\psi}_c$ with $\hat{\dot{\psi}}_c$. Then actions `forw`, `back`, `clock`, `c_clock` and `neut` are evaluated for each tile with a time-step Δt of 0.5s, generating the graph. This choice of Δt is motivated by the need of a sufficiently long time-step to correctly represent the pitch configuration transition, but nonetheless small enough to allow for a faithful incremental form as in Eq. (9). Then, eligible commands are chosen as constant commands with duration comprised between $k_{min} = 3$ and $k_{max} = 5$ time steps. A function $\mathcal{W}$ simulates the presence of on-board sensors: the quadrotor receives a warning signal when within 2.5m of any wall. Hitting a wall or abandoning the restricted state space $\mathcal{S}$ results in a failure. A succesful task consists in reaching 300 iterations without a failure. Each episode is initialized in a random condition:

$$x_0 = 0\,;\ y_0 = 0\,;\ V_0 \in [0.4,\ 0.6]\,;\ \psi \in [-\pi,\ \pi]\,;\ \dot{V}_c \in \hat{\dot{V}}_c\,;\ \dot{\psi}_c \in \hat{\dot{\psi}}_c \tag{15}$$

Either the OM, the PM or a random selector are implemented during execution to complete the algorithm.

1. *Operative metric for quadrotor control*

The values q are initialized as $q_{exp} = 1$, $q_{safe} = 0$, $q_{unc} = -100$, $q_{fat} = -10^6$. A typical behaviour resulting from the application of the metric is shown in figure 5a. In the first instants of flight, the quadrotor is far from the walls. Initially, the controller does not alter the pitch configuration, but instead select actions `neut`, `clock` and `c_clock` repeatedly to move around the room at constant speed. After a few iterations, when the central region of the room have been explored, actions `forw` and `back` are selected as well: it can be noted in the figure that the agent occasionally inverts the direction of flight. When in proximity of a wall, the UAV adopts two strategies to avoid collision. The most common strategy is steering with a costant rate until the collision is avoided. A second strategy, highlighted in figure 5b, consists in changing pitch as to invert direction of flight. This less frequent option is adopted by the controller only in such cases where steering is not a reliable option, e.g. when the quadrotor is headed towards a corner. Simulations show how applying the OM results in a safe flight that avoids collisions and at the same time explores the environment accordingly.



(a) Safe trajectory at almost constant speed.

(b) The controller changes pitch configuration (in green)

Figure 5: Two sample simulations with the operative metric. The black dot represents the starting position of the quadrotor in the room (delimited by black lines). The blue line represent the trajectory. The red dot represent the final position.

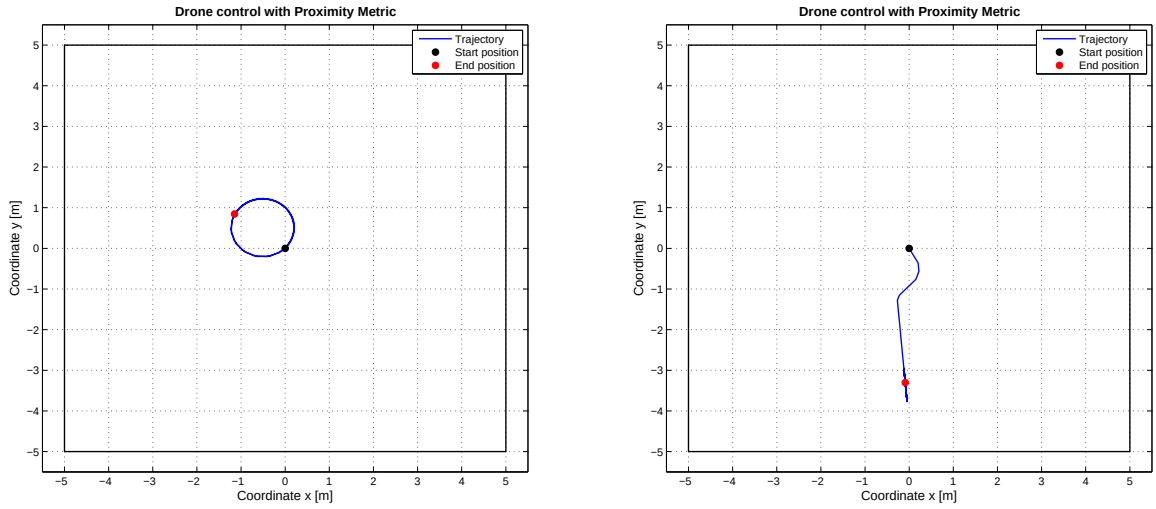
2. *Proximity metric for quadrotor control*

The gain vector $\boldsymbol{\rho}$ is selected as:

$$\boldsymbol{\rho} = (\rho_x \ \rho_y \ \rho_V \rho_\psi \ \rho_\theta) = (5,\ 5,\ 2,\ 1,\ 1) \tag{16}$$

A lower gain is assigned to those components of the state that are immediately accessible from the controller, i.e. ψ and θ. Increasingly higher gains are assigned to V, y and x, aiming to a more cautious controller in those components that are harder to control. The proportional weight η of Eq. (12) is assigned as 0.3. The results showed two different behavior depending on the initial evolution of the system. The steering performance of the quadrotor (indicated by the term $\dot{\psi}_c$) is high enough for it to perform a continuous and steady turn during the whole task (figure 6a). If the controller performs such a turn in the first instants of motion, he will "learn" how to perform a constant turn, and will keep turning indefinitely. This manoeuver will result in a safe flight, but at the cost of halting the exploration. This is the result of the driving concept behind the metric: the controller replicates already encountered conditions. A different behavior stemming from the same concept is shown in figure 6b. In the event that a turn is not performed, the controller will instead make the quadrotor pitch backward in order to reduce the flight speed. When the flight speed is sufficiently low in modulus, the controller will select repeteadly the neutral action `neut` until the quadrotor approaches collision with a wall. The controller will again pitch backward. At this point, two outcomes

are possible: if the speed of the quadrotor is sufficiently high, the controller will not be able to prevent a collision. Otherwise, the controller will manage to invert the direction of flight, and as soon as the quadrotor will start flying in reverse, the controller will resume a neutral pitch and let the system drift with `neut`. As a result, the controller will have "learned" a manoeuver consisting in pitching back and fourth, and from this point onward, will consistently rely on it for the duration of the task.



(a) Safe trajectory with constant turn manoeuver.

(b) Safe trajectory with pitching manoeuver.

Figure 6: Two typical behaviors with proximity metric. The black dot represents the starting position of the quadrotor in the room (delimited by black lines). The blue line represent the trajectory. The red dot represent the final position.

As a final comparison, the mean duration of task observed by random action is of 19.3 iterations, equivalent to 9.65 seconds. With the OM, the controller achieved completion of the task at every run. With the PM, the controller managed completion of the task on 44% of the runs, with minimum duration of 34 iterations and mean duration of 161.

B. Elevator control task

The second task presented in this paper consists in controlling the deflection of the elevator of an aircraft with nominal longitudinal dynamics:

$$\begin{pmatrix} \dot{h} \\ \dot{\theta} \\ \dot{\alpha} \\ \dot{q} \end{pmatrix} = A \begin{pmatrix} h \\ \theta \\ \alpha \\ q \end{pmatrix} + B\delta_e;\ A = \begin{bmatrix} 0 & 300 & -300 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & -0.64 & 0.938 \\ 0 & 0 & -1.568 & -0.879 \end{bmatrix};\ B = \begin{bmatrix} 0 \\ 0 \\ B^{\alpha}_{\delta_e} \\ B^{q}_{\delta_e} \end{bmatrix} \tag{17}$$

assuming constant speed $V = 300$ ft/s. Terms h, θ, α, q and δ_e are deviations from the initial conditions of altitude, pitch angle, angle of attack, pitch rate and elevator deflection. $B^{\alpha}_{\delta_e} = -1.4 \cdot 10^{-3}$ and $B^{q}_{\delta_e} = -0.1137$ are the control coefficients. A change from nominal conditions is introduced by replacing the control matrix B with $B' \in \hat{B} = [1.05 \cdot B, 0.95 \cdot B]$. This can represent either a small malfunction or a unexpected deterioration of the control surface. The goal of the controller is to prevent the aircraft to leave a flight altitude range of $[-80,\ 80]$ft from the initial level flight, while at the same time avoiding a stall by maintaining α in the range $[-15^{\circ},\ 12^{\circ}]$.

Consider now how h and α can be controlled via elevator deflection according to Eq. (17). The main effect of control action δ_e is a pitch acceleration $\ddot{q}$. The dynamic of α is not sensibly affected by δ_e due to the small value of $B^{\alpha}_{\delta_e}$, and is dominated by q and by α itself. Altitude rate $\dot{h}$ depends on the angle $\gamma = \theta - \alpha$, whose derivative can be written as α: $\dot{\gamma} = \dot{\theta} - \dot{\alpha} \cong 0.062 \cdot q + 0.64 \cdot \alpha$. Summarizing, $\dot{h}$ depends on γ,

which is mainly controlled through α. In turn, α can be controlled through q, which is controllable through elevator deflection δ_e. Therefore, this task is an example of low-level control with highly structured, almost hierarchical dynamics.

It will now be shown how to generate the graph for the application of the algorithm. The state space $\mathcal{S}$ is natural with two unbounded states: θ and q, for which respectively $[-\pi/4,\ \pi/4]$ and $[-\pi/2,\ \pi/2]$ are selected as allowed excursions. The grid partition uses 25 intervals for each state, for a total of $25^4 = 390625$ vertices. The action subset is restricted to the four different deflections $\delta_e \in \mathcal{A}_{sub} = \{-4^\circ,\ -2^\circ,\ 2^\circ,\ 4^\circ\}$. A bounding model is obtained by replacing B with $\hat{B}$ in Eq. (17). A function $\mathcal{W}$ simulates the presence of on-board warnings: the agent receives a warning signal when within 30ft of the upper or lower limit altitude treshold, and within 6° of the boundaries of α. The time-step Δt was chosen as 0.1s. Eligible commands have been chosen as constant commands with duration comprised between $k_{min} = 3$ and $k_{max} = 5$ time steps. Violating the constraints on h or α, or abandoning the restricted state space $\mathcal{S}$ results in a failure. A successful task consists in reaching 600 iterations without a failure. Each episode is initialized from starting conditions:

$$h_0 = 0\,;\ \theta_0 = 0\,;\ \alpha_0 = 0\,;\ q_0 = 0 \tag{18}$$

and with a randomly assigned control matrix $B' \in \hat{B}$. Either the OM, the PM or a random selector are implemented during execution to complete the algorithm.

1. *Operative metric for elevator control*

The values q have been initialized as $q_{exp} = 1$, $q_{safe} = 0$, $q_{unc} = -100$, $q_{fat} = -10^6$, as in the previous task. In figure 7, a typical behavior for the controller with the OM is showed. Initially, the controller succeeds in keeping flight path angle γ sufficiently small. However, as the flight height decreases, the controller does not compensate for the altitude loss, because in near-time the predicted states are safe. This is due to the limited scope of the uncertain predictions. As the system approaches the unsafe boundaries of the height range, the commands with the most duration among the candidates (i.e. 5 time steps) become unsafe. The controller is left with the commands of shorter duration as feasible candidates. As the boundaries become nearer, the set of feasible commands restricts even more, to the commands with a duration of 3 time steps. At approximately 13.5 seconds, all near-time predictions become unsafe. In this event, the controller selects a random action, which rapidly leads to a failure of the task.

2. *Proximity metric for elevator control*

Similarly to the previous application, the gain vector ρ is selected as:

$$\rho \ = \ (\rho_h\ \rho_\theta\ \rho_\alpha\ \rho_q) \ = \ (6,\ 4,\ 2,\ 1) \tag{19}$$

that is, the lower the authority of the controller over the state component, the higher the gain assigned. A typical trajectory for the controller with the PM will be now presented. In figure 8, the aircraft starts pitching down, and gradually decreases in altitude. As can be seen in the top of the figure, after a few seconds γ is held almost constant by the controller. This is the result of the formulation of the proximity metric. The controller starts with no visited states. As soon as either a positive or negative flight path angle is experienced, the controller tries to keep the system in this flight condition: this is an example of trailing effect. The higher penalty on deviation in flight altitude keeps this effect under control, limiting the excursion. However, around 27 seconds from the start of the task, the aircraft reaches the boundaries of the region identified as safe. As with the previous example with the OM, in figure 8 the controller is not able to guarantee safety, and switches to a random selection of actions which leads to a violation of the constraints.

Figure 9 shows a different behaviour with the same controller. During this run the controller manages to keep the flight path angle in between $[-1^\circ,\ 1^\circ]$, alternating level flight and mild descent/ascension. This results in a safe flight and in a successful completion of the task; however, only a limited exploration of the environment is achieved during the task. As a final comparison, the mean duration of the task observed by randomly selecting actions for the elevator control task is of 60 iterations, equivalent to 6 seconds. With the OM, the controller achieved completion of the task 15.7% of the runs, with a minimum duration of 58 iterations, and a mean duration of 154. With the PM, the controller managed completion of the task 22% of the runs, with a minimum duration of 129 iterations, and a mean duration of 350 iterations.

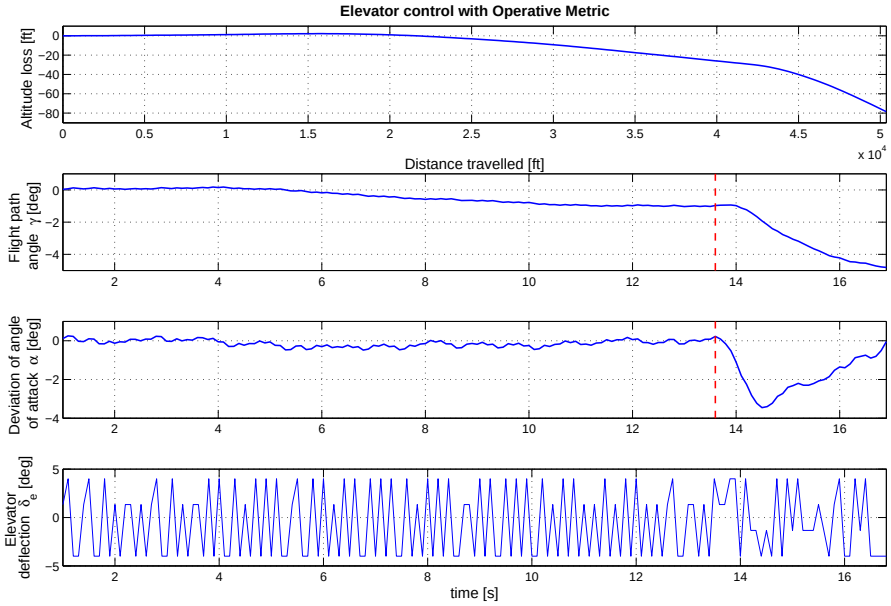


Figure 7: A typical behavior for the controller with the OM during an elevator control task. The altitude loss h with respect to distance traveled is depicted in the top plot. The middle plot shows the change of flight path angle γ with respect to time, while the last plot shows the deviation of angle of attack α with time. The red dashed lines indicate the time of metric failure.

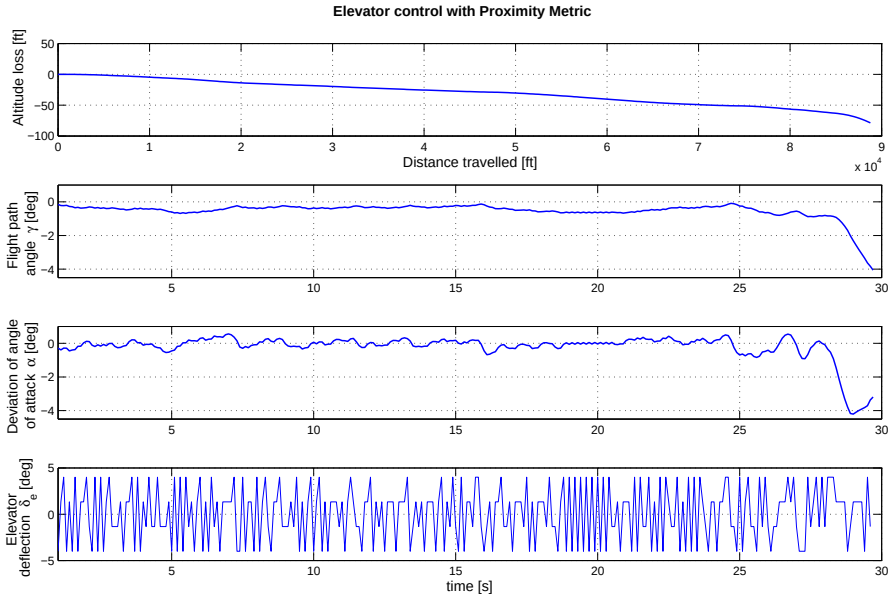


Figure 8: A typical behavior for the controller with the PM. The top plot shows altitude loss h. The middle plot shows the change of flight path angle γ with respect to time, while the last plot shows the deviation of angle of attack α with time. In this example, the controller does not manage to avoid violating the altitude constraint.

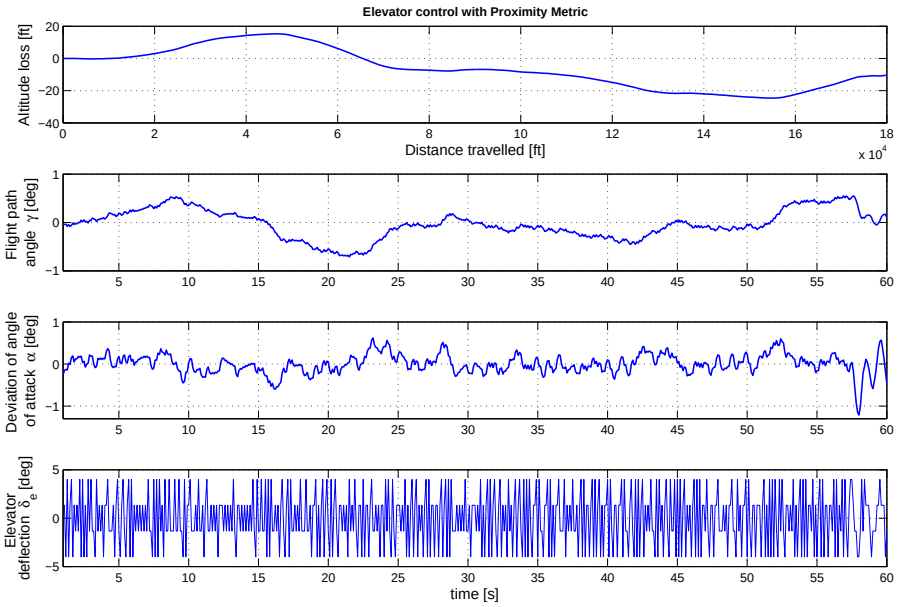


Figure 9: A different episode with the application of the PM. The controller with PM manages to maintain a sufficiently reduced flight path angle γ and to achieve safe flight. However, this results in limited exploration of the environment during the task.

VI. Conclusions and future work

This paper introduced a new approach for Reinforcement Learning exploration of systems with uncertain dynamics and in unsafe environments. The approach revolves around three main elements. The first is the presence of a warning function through which the agent can individuate the fatal states in the environment. The second is an uncertain graph representing the system uncertain model. The system's state space is partitioned via tiling, possibly requiring restricting the space to a bounded subset. A finite representative subset is extracted from among all possible actions available to the agent. The graph can then be generated. The third constituent of the framework is a safety metric, which evaluates candidate commands of the agent at every time-step. Solving this optimization problem yields the action ultimately performed. Two metrics have been proposed: an Operative Metric assigning values to vertices depending on the current belief of safety, and a Proximity Metric computing distances between vertices of the graph and previously visited states. Both approaches have been tested on two different simulated applications: a quadrotor navigation task, for a hybrid, high-level control, and an elevator deflection task, for a low-level control. In the quadrotor task, the Operative Metric was found to be effective in achieving safe exploration, showing intelligent behavior in the selection of the available actions. The proximity metric was not always able to avoid collisions, and resulted in limited exploration. In the elevator task, the operative metric was able to enforce safety only for the first instants of flight. The proximity metric performed better by limiting the rate of altitude loss, achieving longer duration of the task.

The results show that the operative metric enables a reactive controllers that employ the current knowledge of the environment to achieve a good exploration. However the formulation of the metric can be detrimental in tasks with hierarchical dynamics, due to the the limited duration of the available commands, and given the uncertainties in trajectory predictions. Applying the proximity metric has the effect of restraining the evolution of the system. While this results in general in a more efficient prevention of fatal occurrences for system with hierarchical dynamics, as shown in the elevator control task, it can also lead to severely reduced exploration, as in the quadrotor task. In all cases, the two metrics resulted in longer duration of tasks when compared to a random controller.

Summarizing, the results of the simulations indicate that the approach is able to introduce cautious behavior in the agent for high-level and low-level control. However, two considerations are necessary. First,

a controller with either metric still encounters fatal occurrences. A second aspect to consider, especially when considering high-level control, is that the approach can result, as in the application of the proximity metric to the quadrotor task, in a reduced exploration of the environment. Nonetheless, the two applications of the controller with safety metrics indicate that the approach presented in this work is promising, overall increasing safety in unknown environment without relying on an exhaustive prediction of all possible evolutions of the system, relying instead on an approximated modeling of the dynamics. Future work will include the design and evaluation of new metrics and the implementation of such a controller in conjunction with additional elements to promote exploration. A combination of the two metrics seems a promising field of investigation as well.

References

[1]Sutton, R.S., Barto, A.G., *Reinforcement Learning: An Introduction*, MIT Press, Cambridge, MA, (1998)

[2]Coraluppi, S.P., Marcus, S.I., *Risk-sensitive and minimax control of discrete-time, finite-state Markov decision processes*, Automatica, Vol. 35, Iss. 2, pp. 301-309, (1999)

[3]Heger, M., *Consideration of Risk in Reinforcement Learning*, 11th International Machine Learning Conference, Rutgers University in New Brinswick,NJ, (1994)

[4]Hans, A., Schneegaß, D., Schäfer, A.M., Udluft, S., *Safe Exploration for Reinforcement Learning*, ESANN'2008 proceedings, European Symposium on Artificial Neural Networks- Advances in Computational Intelligence and Learning, Bruges, Belgium, (2008).

[5]García, J., Fernández, F., *Policy Improvement through Safe Reinforcement Learning in High-Risk Tasks*, IEEE Symposium on Adaptive Dynamic Programming And Reinforcement Learning (ADPRL), Paris, France, pp. 76-83, (2011)

[6]Moldovan, T.M., Abbeel, P., *Safe Exploration in Markov Decision Processes*, Proceedings of the 29th International Conference on Machine Learning, Edinburgh, Scotland, UK, (2012)

[7]Gillula, J.H., Huang, H., Vitus, M.P., Tomlin, C.J., *Design of guaranteed safe maneuvers using reachable sets: Autonomous quadrotor aerobatics in theory and practice*,IEEE International Conference on Robotics and Automation (ICRA), Anchorage, AK, pp. 1649-1654, (2010)

[8]Gillula, J.H., Tomlin, C.J., *Guaranteed safe online learning of a bounded system*,IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), San Francisco, CA, pp. 2979-2984, (2011)

[9]Mannucci, T., van Kampen, E., De Visser, C.C., Chu, Q.P. *SHERPA: a safe exploration algorithm for Reinforcement Learning controllers*, Proceedings of the SciTech AIAA Guidance, Navigation, and Control Conference, Kissimmee, Florida, (2015)

[10]Henzinger, T.A. *The Theory of Hybrid Automata*, Proceedings of the Eleventh Annual IEEE Symposium on Logic in Computer Science (LICS), pages 278-292, (1996).

[11]Watkins, C.J.C.H., *Learning from delayed rewards*, PhD Thesis, University of Cambridge, England, (1989)

[12]Abbeel, P., Quigley, M., Ng, A.Y., *Using Inaccurate Models in Reinforcement Learning*, Proceedings of the 23rd International Conference on Machine Learning, Pittsburgh, PA, (2006)

[13]Moore, R.E., *Interval Arithmetic and Automatic Error Analysis in Digital Computing*, Ph.d. Dissertation, Department of Mathematics, Stanford University, Stanford, California, Published as Applied Mathematics and Statistics Laboratories Technical Report No. 25, (1962)

[14]Freichard, T., Asama, H.,*Inevitable collision states. A step towards safer robots?*, proceedings of IEEE International Conference on Intelligent Robots and Systems (IROS), pp. 388-393, (2003)

[15]Sutton, R.S., *Generalization in reinforcement learning: Successful examples using sparse coarse coding*, in Tesauro, G., Touretzky, D., Leen, T., eds.: Advances in Neural Information Processing Systems 8, Cambridge, MA, MIT Press, (1996)

[16]Whiteson, S., Taylor, M.E., Stone, P. *Adaptive Tile Coding for Value Function Approximation*, AI Technical Report AI-TR-07-339, University of Texas at Austin, (2007)

[17]Lin, S., Wright, R., *Evolutionary Tile Coding: An Automated State Abstraction Algorithm for Reinforcement Learning*, in Abstraction, Reformulation, and Approximation, volume WS-10-08 of AAAI Workshops, AAAI, (2010)

[18]Morari, M., Garcia, C.E., Prett, D.M., *Model predictive control: theory and practice*, in Automatica 25 (3), pp.335348, (1989)