Safe Online Robust Exploration for Reinforcement Learning Control of Unmanned Aerial Vehicles

Mannucci, Tommaso

DOI
10.4233/uuid:dbaf67cc-598c-4b26-b07f-5d781722ebfd

Publication date
2017

Document Version
Final published version

Citation (APA)
Mannucci, T. (2017). *Safe Online Robust Exploration for Reinforcement Learning Control of Unmanned Aerial Vehicles*. [Dissertation (TU Delft), Delft University of Technology].
https://doi.org/10.4233/uuid:dbaf67cc-598c-4b26-b07f-5d781722ebfd

Safe Online Robust Exploration for Reinforcement Learning Control

of Unmanned Aerial Vehicles

Safe Online Robust Exploration for Reinforcement Learning Control

of Unmanned Aerial Vehicles

Proefschrift

ter verkrijging van de graad van doctor
aan de Technische Universiteit Delft,
op gezag van de Rector Magnificus prof. ir. K.C.A.M. Luyben,
voorzitter van het College voor Promoties,
in het openbaar te verdedigen
op donderdag 12 oktober 2017 om 15:00 uur

door

Tommaso MANNUCCI

Master of Science in Aerospace Engineering,
University of Pisa, Italië,

geboren te Pisa, Italië

Dit proefschrift is goedgekeurd door de promotor:

Prof. dr. ir. M. Mulder

Copromotor:
Dr. ir. E. van Kampen

Samenstelling promotiecommissie:

Rector Magnificus,	voorzitter
Prof. dr. ir. M. Mulder,	Technische Universiteit Delft, promotor
Dr. ir. E. van Kampen,	Technische Universiteit Delft, copromotor
Onafhankelijke leden:	
Prof. dr. M. Beetz,	Universität Bremen
Prof. dr. ir. R. Vingerhoeds,	Institut Supérieur de l'Aéronautique et de l'Espace, Toulouse
Prof. dr. D. G. Simons,	Technische Universiteit Delft
Prof. dr. ir. M. J. T. Reinders,	Technische Universiteit Delft
Prof. dr. ir. J. A. Mulder,	Technische Universiteit Delft

Keywords: Unmanned Aerial Vehicles, Reinforcement Learning, Safe Exploration, Hierarchical Reinforcement Learning

Printed by: Ipskamp Printing

Front & Back: Designed by Giulio Zannol.

ISBN 978-94-028-0762-2

An electronic version of this dissertation is available at
`http://repository.tudelft.nl/`.

Temer si dee di sole quelle cose
c'hanno potenza di fare altrui male;
de l'altre no, ché non son paurose.

Of those things only should one be afraid
that have the power of doing injury;
not of the rest, for they should not be feared.

Beatrice, Inferno II, 88-90

Summary

Safe Online Robust Exploration for Reinforcement Learning Control

of Unmanned Aerial Vehicles

Tommaso **Mannucci**

In recent years, the aviation domain is witnessing an unprecedented surge of interest in unmanned aerial vehicles (UAVs). With the advancement of miniaturized and low-cost hardware, ranging from circuits to sensors, UAVs are steadily becoming cheaper to produce and are rapidly improving their performance and endurance. As a result, "drones" have now entered the recreational market as affordable toys and reliable working tools. Furthermore, various companies are investigating into adopting drone fleets in the near future for cost-effective services, such as deliveries and distribution.

Conversely, control design for UAVs still relies heavily on classic control techniques, such as PIDs or robust controllers. Indeed, these controllers are reliable with respect to model inaccuracies, which are very common in UAVs and in particular in micro air vehicles (MAVs). One practical drawback, however, is that these techniques require considerable effort for gain tuning, testing, and modeling during the design stage. The prospect of entirely autonomous UAV tasks, without the supervision of a human operator, constitutes a further challenge for these classical controls, and is likely to further and significantly increment the burden of control design.

In this perspective, reinforcement learning (RL) has the potential to overcome these difficulties. RL is a branch of machine learning that mimics animal learning: an *agent* repeatedly interacts with its *environment* through *actions*, receiving each time a *reward*. This indicates the immediate goodness of its choice, according to a reward function provided by the designer. The agent goal is then to collect the maximum amount of discounted reward, which constitutes an optimal *policy*. The strength of this *exploration* procedure is that the agent can learn autonomously, adaptively, and model-independently.

When performing RL exploration in the application of a flying vehicle, it is of paramount importance that this exploration is performed safely. The agent must identify unsafe actions, e.g., those that might result in a collision, without actually applying these actions. This is the *challenge of safety*. It would in principle be possible to guarantee safety by learning in a safe or simulated replica of the actual environment, in which unsafe actions are allowed; however, the policy learned this way might not be safe within the actual environment, if there are discrepancies and uncertainties in the replica. This constitutes the *challenge of robustness*. Furthermore, assuming the safety of an action can be evaluated online, this assessment must be computationally simple enough for the agent to do the assessment while controlling the UAV in real time. This is the *challenge of online efficiency*.

It is clear that these three cardinal challenges must be overcome before UAVs and MAVs can thoroughly benefit from the advantages of RL. The goal of this dissertation is to investigate these problems of online, safe, robust exploration for UAV platforms, and to develop potential solutions in accordance with the properties of adaptability, autonomy and model independence of RL.

Safety is the first and by far the most compelling challenge for the agent. To simplify the problem, it is postulated that the unsafe actions are those that cause the environment to transition to an element of the fatal state space, which is postulated to be unknown but time-invariant. In absence of an a-priori known safe policy, or of a human teacher, two key capabilities are deemed necessary to avoid fatal transitions. The first is *risk perception*, which takes the form of an additional feedback from the environment to the agent, and that informs the agent whether or not an element of the fatal state space is within a predetermined neighborhood of the current state. The second is a *bounding model* which overestimates the future transitions of the environment given the agent actions.

These two capabilities form the central strategy of the *Safety Handling Exploration with Risk Perception* Algorithm (SHERPA), developed in this thesis. This "safety filter" is placed between the agent and the environment. SHERPA allows only actions guaranteed not to cause a fatal transition, and that can be followed by a *backup*, i.e., a sequence of action that causes the environment to transition to a safe neighborhood of a previously visited state. In case the action proposed by the agent is refused by SHERPA, the agent is queried to propose a different one, until either SHERPA approves, or a time limit is reached, at which point SHERPA executes a pre-approved backup. By doing so, SHERPA provides the agent with a safety assessment that is autonomous and based on online experience. SHERPA is validated on a simplified quadrotor task, where it is found to be more effective than a competitor method.

This dissertation further develops the SHERPA strategy into OptiSHERPA. This successor algorithm turns safety assessment into an optimization problem, choosing the safest action among a restricted set of candidates. Additionally, an evasion metric is added so that, if risk is perceived, the agent adopts the action that is least likely to cause a fatal transition. OptiSHERPA is tested in a control task for a fighter aircraft with linearized model.

The second challenge of *online efficiency* is addressed in this dissertation by using *graphs* during safety assessment. For example, SHERPA utilizes the bounding model dynamics to predict the state trajectory given a sequence of actions: the faster the computation of these trajectories, the higher the number and the feasibility of the assessments of proposed actions in real-time exploration.

The graph is generated prior to exploration according to three steps. First, an arbitrary subset of the state space is selected as the *operational envelope* of the agent. Second, the state space, the set of actions, and time are discretized. Third, the bounding model is executed to create edges connecting the vertices of the graph. The result is an hypergraph that overapproximates the bounding model dynamics. The hypergraph is stored in the form of look-up matrices, replacing the online computation of transitions with index checking, which has a lower complexity than applying the dynamics of the bounding model itself. Furthermore, an opportune state discretization is introduced in the form of an evenly spaced tiling. As a result, the graph generation is reduced in complexity and made feasible for online exploration as well.

The graph formulation is implemented within the SHERPA strategy by defining two graph-based *safety metrics*. These assign to each vertex of the graph a weight that corresponds to its safety, according to the specific metric. Safety assessment is then turned into an optimization problem as per the OptiSHERPA strategy. The metrics are implemented in a simplified quadrotor simulation, as well as in aircraft control via elevator deflection. The *operative* metric is found to be more effective in tasks where the insurgence of risks can be more easily predicted, such as the quadrotor task. For more complex ones, as the elevator deflection task, the *proximity* metric is found to be more effective by constraining the evolution of the system with time.

Assuming the operational envelope does not contain fatal states, *graph pruning* is introduced to perform safety assessment of the agent's entire policy. Edges that violate the envelope are removed from the graph, together with vertices that, as a result, have no outbound edges. All policies that are compatible with the pruned graph are therefore safe; however, the opposite is not true due to the uncertainty of the graph. As with graph generation, and depending on the refining of the state discretization, pruning can be made feasible for online exploration of envelopes that are moderately time-varying, as simulated in several MAV corridor tasks.

Uncertainty in a replica of the environment results in the challenge of *robustness*. The more the environment is uncertain, the more the available model must overestimate the actual dynamics to still be bounding, and the more refined the graph discretization must be to contain the overapproximation of trajectories. In this thesis, this problem is mitigated by implementing *Hierarchical Reinforcement Learning* (HRL). By abstracting the state space, by embedding design knowledge, and by constraining the state of discoverable policy, HRL is found to contribute to both the challenge of robustness and the challenge of safety. The novel combination of HRL methods for the scope of Safe Reinforcement Learning is presented as Safe Hierarchical Reinforcement Learning (SHRL).

The SHRL method of *Virtual Safety Training* (VST) proposed in this thesis consists of three steps. First, the original state space is transformed, via an arbitrary *projection function*, in such a way that at least one projected space is independent to and/or relative from the others. This allows one to reduce the complexity and the uncertainty of the environment. In the second step, a belief set representing the possible projection of the fatal state space is adopted. Then, an initial policy for the agent is learned off-line in the "virtual" projected learning space, once for each belief of the set. The efficacy of this method is found to depend on the exhaustiveness of the belief set and on the uncertainty in the projected bounding model. Tested in an MAV goal finding task within a cluttered environment, the strategy is found to be safer, even with an unsophisticated model and a simple belief set, than a non-hierarchical policy learned within the actual environment.

As a second and final SHRL method, Vertex Classification (VC) is introduced. This method integrates all previous contributions, such as safety optimization, graph formulation of the dynamics, and state projection, in order to solve the problem of safe exploration sequentially. First, an operational envelope for the projected state space is defined, and a graph is created using the projected dynamics. Those edges that lead to a violation of the envelope are ordered according to their level of undesirability, which in turn is used to compute two sets of weights, the *levels* and the *coefficients*, for each vertex and per each violation. Finally, by assigning to each violation an *intensity*, the weights are used to estimate the safety of the edges of the graph. The *safest policy* of VC, validated in the same MAV task as VST, prevents all collisions when tested with different model realizations and with different obstacle dispositions. Furthermore, it is found to share resemblances with potential field methods, such as getting stuck in local minima between goal and obstacles.

As a final consideration concerning the main objective of achieving safe, online, robust RL exploration for UAVs, this dissertation contributes to the state-of-the-art by providing several methods which mitigate each challenge individually, as well as hybrid algorithms that address multiple challenges simultaneously. In order to address these challenges, explicit and clear assumptions for the application of the proposed methods are provided; these assumptions notwithstanding, the methods are developed to adhere to the principles of autonomy, adaptability and model independence of RL, as much as the problem of safe exploration allows. Nonetheless, several points of further development and improvement are put forward within this dissertation addressing both the machine learning and the UAV operator communities.

Contents

1

Introduction

1.1. Emergent automation in society, industry and aerospace

Automation is widely accepted as one of the driving innovations in today's technological growth of the aerospace sector. Generally speaking, an increase in automation is associated with a reduction in costs due to the replacement of both skilled and unskilled labor, paired with an improvement of performance as well as working conditions. These incentives have up to now encouraged an increase of automation in all economic sectors. For example, several companies[1] are currently discussing and competing upon developing self-driving cars. This does not only represent a goal for the automotive industry, but is a hot topic of public debate as well.

Aerospace is no exception to this trend. Since their first prototypical versions, dating back to the first half of the 20th century, military unmanned aerial vehicles (UAVs) boasted increasingly advanced sensory equipment and computing capabilities, and became able to perform more and more complex tasks. Circuits, batteries and sensors of recent years have increased in performance for a fraction of the cost and of the weight. As a result, UAVs have entered into civil use after decades of military exclusivity, and are now a growing sector in the recreational mass market. Notable investments are being made as of today by major companies, such as Amazon[2] and Walmart[3], to employ a fleet of UAVs for delivery and distribution. As a result, it is widely accepted and foreseen that, regulatory issues aside, aerospace as a whole will become more and more automated.

[1] McCurry, J., "Honda in talks over self-driving cars with Alphabet's Waymo", 2016, accessed Feb. 2017, https://www.theguardian.com/technology/2016/dec/22/honda-in-talks-over-self-driving-cars-with-alphabets-waymo

[2] Kharpal, A., "Amazon's latest drone delivery idea involves parachuting parcels into your backyard", 2017, accessed Feb. 2017, http://www.cnbc.com/2017/02/17/amazons-latest-drone-delivery-idea-involves-parachuting-parcels-into-your-backyard.html

[3] Abrams, R., "Walmart Looks to Drones to Speed Distribution", 2016, accessed Feb. 2017, https://www.nytimes.com/2016/06/03/business/walmart-looks-to-drones-to-speed-distribution.html

However, improving automation is by itself a challenge when confronted with traditional control design in aerospace. Consider for example classical Proportional-Integral-Derivative (PID) controllers, which even today are an important component of many applications. PID controllers are rather easy to implement, and are relatively robust to modeling error. Nonetheless, they require a significant effort when tuning the gain values. Furthermore, a model should be provided to facilitate an initial, tentative tuning; however, UAVs often have nonlinear, unstable dynamics that are difficult to model. In the extreme case of micro air vehicles (MAVs), modeling procedures are still experimental and unstandardized [1–3].

Even though more advanced control has improved over traditional control techniques, considerable design effort is still needed to meet specifications, in the form of modeling, testing and performance assessment. For example, robust controllers improve on PID controllers by addressing uncertainties in the original model, but ultimately require from designers an amount of effort comparable to PID tuning. Considerable amount of insight, tools, money and time is still a necessity for control design. As the market expectation for more complex, autonomous UAVs able to perform multifaceted tasks grows, this burden is likely to increase in the years to come. What if a method existed to obtain controllers able to perform complex and diverse tasks with minimal effort from the designer?

1.2. Reinforcement learning

Machine learning methods, and in particular reinforcement learning (RL) [4], have the potential to much better address the challenges of complex control tasks than traditional control theory. RL is a bio-inspired decision making approach that mimics animal learning. According to behavioral psychology, animals learn on the basis of stimuli: after an animal makes a decision and interacts with its environment, e.g., picks and eats a fruit, it immediately receives a corresponding reinforcement stimulus, which conveys and represents the immediate benefit of that behavior, e.g., feeling satiated after eating. If the result of the action is beneficial within a situation, the corresponding behavior is reinforced, i.e., it can be observed to be opted for more often; conversely, if the result is not beneficial, the animal will refrain from following said behavior in the future. The more a behavior is reinforced this way, the more it is likely to be beneficial for the animal (for that given situation), and the more the animal consequently adopts it.

1.2.1. Working principle

RL can be used to discover policies for UAVs, in a learning process that is autonomous, adaptive to environmental conditions, and that does not need a model of the vehicle. Figure 1.1 gives a representation of the RL scheme. Here, the *agent* represents the controller, i.e., the embodiment of the decisional process of the UAV, and is equivalent to the animal itself. The *environment* represents the set of all the relevant world features, according to the task settings and specifications. It is equivalent to the actual natural environment of the animal, but it also includes the internal conditions of the UAV, e.g., residual battery life. The state s of the

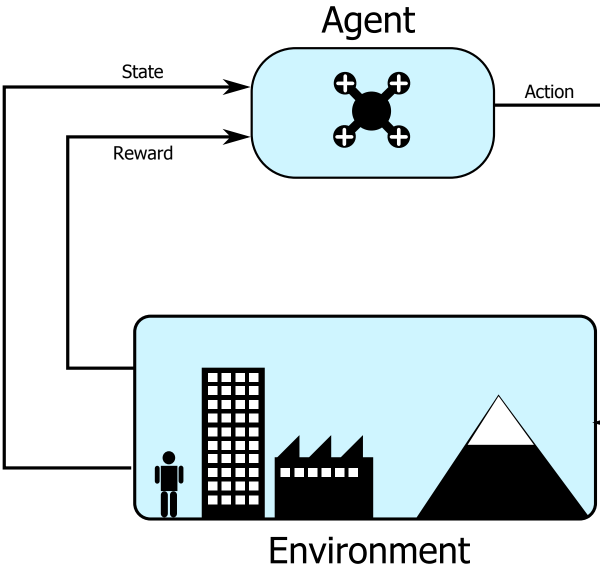


Figure 1.1: Working principle of RL. The agent interacts with the environment through actions. Subsequently, it observes the new state and the reward corresponding to the state transition.

environment is therefore the decision space of the agent. When the agent applies an *action* a to the environment, it either causes a transition from its current state s_k to a different state s_{k+1}, or the system does not change, i.e., $s_{k+1} = s_k$. In both cases, the agent receives a *reward* r, a real valued signal which indicates the immediate benefit of the transition, and which is equivalent to the aforementioned reinforcement stimulus. High reward indicates that the transition is beneficial to the agent. Conversely, low reward means that the transition is disadvantageous. Rewards are sometimes replaced in the literature by *costs*, so that a high cost is equivalent to a low reward, and vice versa. For finite, discrete environments, the above mechanism of *state-action-state* transition and the corresponding reward assignment are modeled as a Markov decision process (MDP) (see Figure 1.2). The *Markov property* of MDPs guarantees that state transitions and rewards depend uniquely upon the current state and action.

As for a foraging animal the best behavior is the one that maximizes the positive reinforcement, a policy $a = \pi(s)$ is optimal if it maximizes the total discounted sum of future rewards, or alternatively, if it minimizes the total discounted sum of future costs. The discount factor $\gamma \in (0, 1]$ determines how much future rewards can affect current behavior. This sum is expressed implicitly in the value function

$$V^{\pi}(s_0) = \sum_{k=0}^{\infty} \gamma^k r(s_k, \pi(s_k), s_{k+1}), \tag{1.1}$$

which essentially indicates how desirable the state s_0 is for the agent, according

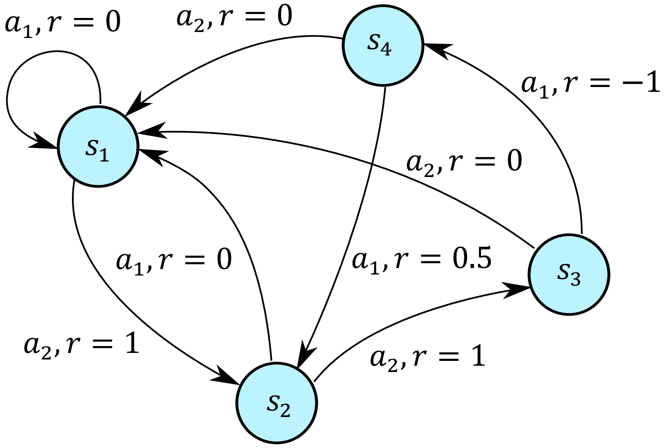


Figure 1.2: An example of a deterministic MDP with four states s_1, s_2, s_3 and s_4 and two actions a_1 and a_2. Values of reward range from $r = -1$ to $r = 1$.

to policy π[4]. According to Bellman's Principle of Optimality [5], the optimal value $V^*(s)$ of a state must follow the Bellman System of Equations:

$$\forall k, V^*(s_k) = \max_{s_{k+1}} r(s_k, a_k, s_{k+1}) + \gamma V^*(s_{k+1}) \tag{1.2}$$

A common approach to solve Eq. (1.2) during learning is to use temporal difference (TD) methods, which consist in subdividing the Bellman equivalence in multiple updates as:

$$V(s_k) \leftarrow V(s_k) + \alpha(r + \gamma V(s_{k+1}) - V(s_k)), \tag{1.3}$$

with a learning rate α whose role is to facilitate convergence. Once the value function converges, the optimal policy is simply to choose the action for which $V(s_{k+1})$ is maximum. This thesis will focus mainly on the application of TD methods, but other approaches to solve Eq. (1.2) exist, e.g., Dynamic Programming (DP) or Monte Carlo methods (Figure 1.3).

A final distinction within RL exists between online and off-line learning. Off-line learning implies that the agent owns an amount of information about the problem MDP, i.e., a model of the environment and of the reward. In this case, it can tentatively learn a policy without the strict need of interacting with the environment and observe the reward. Conversely, learning is said to be online when the agent has the necessity of interacting and observing the environment in order to learn a policy. While this distinction is clear, RL algorithms are not stricly confined between learning off-line or online, but can transition from one to the other and even alternate.

[4] In the event that either the state transition or the reward function are stochastic, the value of state s_0 is defined as the expected value of the right-hand term of Eq. (1.1).

1.2.2. General properties and advantages

RL possesses the three properties of *autonomy*, of *adaptability*, and of *model-independence*. The first property, *autonomy* of learning, directly derives from the reward function of Eq. (1.1). Differing from other machine learning methods, such as supervised learning (see Figure 1.3), the main if not the only duty of the teacher is to design the reward function, whose sum the agent will proceed to maximize. Choosing a reward function for which the resulting behavior satisfies its intended purpose requires expertise and insight; however, this methodology of learning is extremely efficient when the final goal of the policy is known, for example rescuing survivors of earthquakes, but there is no a-priori indication on how to achieve it, e.g., the location of the victims is unknown.

The second property, *adaptability*, comes from updating the value function. When the function has converged, the term that multiplies α, called the TD error, will be approximately zero. If the environment changes, e.g., a goal is moved from its original position, this change will be reflected in the reward function, so that the TD error will no longer be equal to zero. Therefore, any meaningful change in the environment will renew the optimization process, with the final result of adapting the policy to the new environmental conditions.

The definition of optimal policy as the one for which $V(s_{k+1})$ is maximum loses its purpose if a model formulation of state transitions is not available, or is not sufficiently predictive. As previously stated, this is the case for several UAVs and MAVs for which obtaining a precise model of the dynamics is difficult or infeasible. A fundamental step towards solving this inconvenience was the introduction of *model-free* algorithms, such as the very successful Q-learning [6]. These methods replace $V(s)$ with an action-value function $Q(s,a)$ which maps state-action pairs to the maximum possible return obtainable after taking action a in state s. The revised update rule

$$Q(s_k,a_k) \leftarrow Q(s_k,a_k) + \alpha\,(r + \gamma \max_{a_{k+1}} Q(s_{k+1},a_{k+1}) \; - Q(s_k,a_k)) \tag{1.4}$$

guarantees the aforementioned definition upon convergence. Several other model-free methods are present in literature, such as SARSA [7].

The third property, *model-independence*, is exhibited by model-free RL methods. These methods can be applied when the vehicle dynamics are only approximately known, and even when they are entirely unknown. These methods do not need to predict the successor state when taking a certain action; instead, they provide the optimality of the state-action pair itself. The optimal policy available to the agent can therefore be learned regardless of prior model knowledge, and can adapt to alterations in both the environment and the transition function. This is especially useful in aerospace applications, where some dynamics and interactions might be loosely modelled or not modelled altogether, e.g., aerodynamic couplings, and where operating conditions change with time, e.g., due to fuel consumption.

Summarizing, RL methods can autonomously find optimal policies for complex tasks, regardless of whether an initial policy is provided for the agent, and can

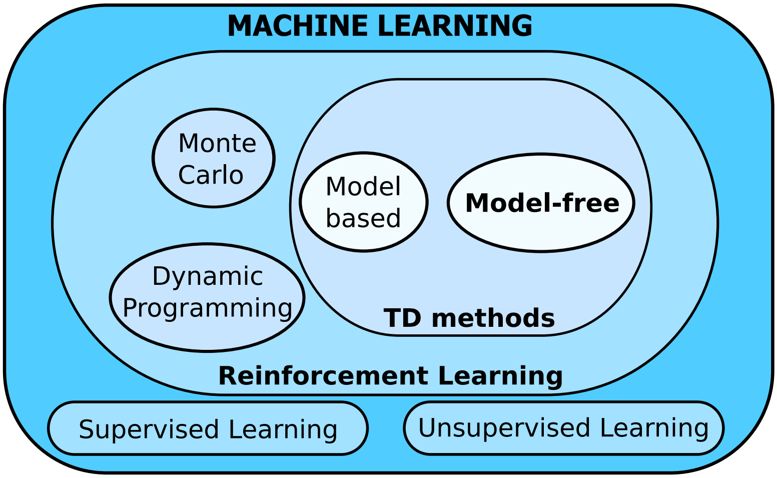


Figure 1.3: Taxonomy of the different methodologies of Machine Learning, reinforcement learning, and TD-methods.

accommodate uncertainties and changes in both the environment and the vehicle itself.

1.3. Challenges of reinforcement learning for UAVs

In Section 1.2, the main advantages and properties of RL methods have been illustrated, as well as the reasons why these methods are promising and appealing for both future automation and the aerospace field. However, RL for UAVs and MAVs comes with three main challenges that must be addressed in order to fully benefit from these methods, as it will be explained in detail in this section.

Challenges

1. **Safety:** *Online reinforcement learning, due to its inherent trial-and-error nature, cannot be safely applied to real-life UAVs.*
2. **Robustness:** *Off-line reinforcement learning using a simulated, artificial or supervised environment is impractical and not robust with respect to errors in replicas of the environment.*
3. **Online efficiency:** *Efficient and low-complexity control algorithms are required to perform safe and online exploration with UAVs.*

1.3.1. The challenge of safety: infeasibility of blind search for aerospace vehicles

The one caveat of the properties discussed in Section 1.2.2 is that significant interaction, in terms of state-action pairs performed by the agent, is necessary to reach convergence of the algorithm. This is self-evident from Eq. (1.3), since $V(s_k)$ updates for visited states only and in accordance with functions r and V which depend on states-action-state trials s_k, a_k, s_{k+1}. The impact that this requirement has on the applicability of RL varies, but it is well known that, as the size of the learning problem increases, the amount of interaction samples to convergence increases exponentially: a problem which Bellman referred to as the *curse of dimensionality* [5].

How to avoid the curse has been a primary focus since the early developments of RL. Various strategies, such as TD(n) methods, eligibility traces, tile coding, Adaptive Dynamic Programming, among others, attempted to reduce the number of necessary trials, both simulated and in real life, needed to obtain at least a satisfactory policy. That notwithstanding, a RL agent needs to visit a significant portion of the learning space before learning can be considered complete. At the very beginning of this learning phase, called *exploration*, the agent often has no or very limited knowledge of the task, and is forced to repeatedly try random actions, in a process known as *blind search* [8], during which it will most likely perform poorly.

When aerospace vehicles perform online exploration, an agent in blind search exploration might not only have suboptimal performance, but can also risk damaging the vehicle itself or its surroundings, due to unsafe tentative actions. This discourages the use of RL for learning in real-life tasks. Safety considerations apply to other systems than RL UAV agents. E.g., in the aforementioned example of self-driving cars, safety is a primary question and an open debate[5] exists about the participation of such vehicles in public traffic. For RL agents, this problem is even more crucial because of the trial-and-error nature of the method.

In general, blind search exploration is detrimental and unsafe for all systems or platforms that learn online, i.e., from real-life experience, and a) are fragile or vulnerable with respect to the intended operations, b) can harm persons and objects nearby, c) are expensive or inconvenient to replace/repair. Typical aerospace vehicles belong to all three categories, being at risk of collision and crashes, achieving considerable velocities, having rotating parts, and being relatively expensive.

1.3.2. The challenge of robustness: detrimental effects of direct learning in simulated, artificial or supervised environments

Since online RL is not safe, it would be highly preferable to perform learning in an off-line setting. The inevitable problem in this case is that a replica of the actual environment must be provided to the agent. Two choices are possible: either letting

[5] Solon, O., "Why self-driving cars aren't safe yet: rain, roadworks and other obstacles", 2016, accessed Feb. 2017, https://www.theguardian.com/technology/2016/jul/05/tesla-crash-self-driving-car-software-flaws

the agent learn in a physical, but controlled and safe environment, or letting the agent interact with a simulated approximation of the actual environment.

Artificial environments, such as a controlled lab, could be explicitly tailored to make learning safer. However, doing so would reduce the benefits of RL as a learning method; moreover, it could introduce several learning biases in the final learned policy. In order to secure the agent, a risk assessment must first be formulated; then all possible dangers must be either removed from the environment, or the agent must be supervised and stopped when approaching such dangers. In the first case, a severe discrepancy is introduced between the real environment, where ultimately the agent must perform, and the learning one. This "reality gap" not only affects the agent's performance, but also its safety. The same happens when the agent learns under supervision, with the additional disadvantage that the learning is effectively not autonomous but dependent on the supervisor himself. Furthermore, the presence of the supervisor can involuntarily become part of the agent's learning [9].

As an alternative, simulated environments can be considered autonomous and therefore do not require the presence of a supervisor. During simulations, the agent can safely attempt dangerous actions and directly learn from them without any real risk. However, contrarily to a lab environment, learning in simulations also requires a sufficiently accurate knowledge of the model of the vehicle. If the level of accuracy is insufficient, the fidelity error can be considerable, and the learned policy might be dangerous when applied to the real-life system. This is a serious drawback since, as previously mentioned, obtaining high fidelity models for UAVs, and especially for MAVs, is difficult. In conclusion, learning off-line as an alternative to on-line learning is not an attractive option, since the resulting policy is unlikely to be robust to the fidelity error in the environment (for labs) or in the model (for simulations).

As a last remark, it must be noted that faithfully reproducing the task environment is not always an option: the environment might be partially unknown, or might be evolving with time. In conclusion, profitable learning must inevitably happen online, in the real environment, so that the agent learns its actual task, as well as the corresponding dangers.

1.3.3. The challenge of online efficiency: computational limitations of UAV online learning

In a safe environment, there is no restriction to which actions the agent can attempt when exploring either online or off-line. Indeed, even performing "bad" actions is valuable, as it will hopefully teach the UAV not to perform such actions in the future. This ceases to be true in case of dangerous environments, where the agent must be prevented or greatly discouraged from attempting dangerous actions even once. It is therefore clear that the decisional process of action selection of the agent must include at least one validation step to ensure that it will not incur danger. Therefore, regardless of how this validation is performed, unsafe environments tend to require more cumbersome computation of actions.

Unfortunately, during online learning the time required for computation has an effect not only on performance, but also on safety. In many practical examples considered in literature, computational complexity is not a major issue, and it is a common assumption that the agent has enough time to observe the state, adapt some inner logic, modify its policy, and apply the new corresponding action. Even where computational complexity is relevant, as in planning algorithms [10], this is usually for the sake of performance, rather than for safety; e.g., a wheeled robot exploring a room can safely remain in position and do nothing while the appropriate action is computed.

UAVs and MAVs operating in dangerous environments represent a very different setting. First of all, these vehicles are highly dynamic and more prone to instability and external disturbances. For example, a UAV must be able to counteract a gust or a similar disturbance with an appropriate action, but at the same time verify the overall safety of said action. Some UAVs, similarly to wheeled robots, are able to remain in place by hovering, e.g., quadrotors. However, other UAVs, such as those with fixed wings, cannot hover. Furthermore, UAVs that can hover require dedicated controllers in order to do so, which in turn need a model, and which are not easy to design in presence of disturbances, e.g., wind. Therefore, an airborne agent's policy must be more reactive to the insurgence of risks compared to that of a ground agent. Furthermore, the stricter requirements on weight and endurance that UAVs and MAVs possess when compared with other autonomous agents limit the computational capabilities of these platforms. As a result, action computation must be as simple as possible for UAV RL agents exploring online in dangerous environments.

1.4. Research goal, methodology and scope

This section illustrates the objectives, the contributions and the limitations of the dissertation.

1.4.1. Research objectives

Research objectives

The goal of this thesis is to investigate the problem of online, safe, robust exploration for aerospace platforms, and to develop potential solutions to the problem of unsafe blind search in accordance with the properties of adaptability, autonomy and model independence of reinforcement learning.

In order to accomplish these objectives, the following research steps are taken:

- The first step is to assess how the properties of aerospace vehicles alter the effectiveness of pre-existing safe exploration algorithms, and find any additional requirements (Chapter 1);

- The second step is to individuate a strategy that can address the challenge of **safety**, in order for the agent to explore online. The strategy must be compliant with the aforementioned requirements (Chapter 2);

- The third step is to devise methods which are compatible with the previous strategy and that have low computational complexity, in accordance to the challenge of **online efficiency** (Chapter 3);

- The fourth step is to solve the challenge of **robustness**, so that the above strategies are applicable even in the presence of uncertainties, errors and missing information within the environment (Chapter 4);

- As a fifth step, the results obtained in the previous steps must be merged into one or more unified methods that can address **all of the above challenges** (Chapters 5 and 6).

1.4.2. Research contributions

Contributions

This research contributes with new exploration methodologies that result in online reinforcement learning for UAVs and MAVs being safer and more robust to model uncertainties in a computationally efficient way.

This research promotes the field of safe exploration for UAVs with the following contributions:

- Two safety-filters are developed, the Safety Handling Exploration with Risk Perception Algorithm (**SHERPA**) and its successor **OptiSHERPA**, that address the challenge of **safety** and improve over existing algorithms by relaxing the conditions on the existence of a predefined backup policy by explicitly accounting for the UAV dynamics and by estimating the dangerous states;

- **Graph Pruning**, a novel representation of the feasible policy set obtained by enforcing a predefined *operational envelope*, which addresses the challenge of **online efficiency**;

- A novel hierarchical method coined Virtual Safety Training (**VST**) that addresses the challenge of **robustness** by projecting the state space in order to allow off-line learning in an abstracted and simplified environment.

Additional contributions are given by two hybrid methods:

- A graph-based method with **safety metrics** that combine the risk perception of SHERPA with the computational efficiency of graph methods;

- Vertex Classification (**VC**), which combines risk perception, state projection and graph representation of policies, which results in safe navigation for UAVs by exhibiting potential field-like behavior.

1.4.3. Scope and limitations

In order to reach the research objectives stated in Section 1.4.1, this dissertation adopts the following assumptions and scope restrictions.

Convergence of learning: In RL, if the agent is prevented from exploring some regions of the learning space, i.e., from attempting some state-action pairs, the convergence and optimality of the learned policy can be affected. This is related to the so-called *exploration-exploitation dilemma* [4, 11], which is the conflict between exploiting the best known action or exploring new, possibly better actions. The goal of the methods presented in this dissertation is to provide safety to the exploring UAV agent, rather than proving convergence or optimality. It will be assumed that the final learned policy might be suboptimal only in the event that learning the optimal policy requires attempting possibly unsafe actions during learning.

Stability: Stability is a fundamental requirement in the fields of control and of aerospace. In terms of RL agents, it is rare to see unstable converged policies, as they result in behaviors that are penalized by the reward function. However, it is possible that not yet converged policies might be unstable. In this dissertation, the stability of converged or temporary policies is not directly discussed. Nonetheless, it is assumed that RL exploration that might lead to instability of the vehicle will be prevented, if this instability is a possible cause of danger.

Validation methodology: In engineering, using real-life testbeds proves the validity of an algorithm for a specific application; however, it is common in RL literature to employ simulations, which allow for a better understanding of the agent's behavior under different learning approaches. This is due to the absence of factors external to the learning process, such as noise or hardware malfunctions, and due to the increased amount of learning episodes that can be performed. For these reasons, as well as for reducing the complexity of the research, the methods presented in this dissertation are investigated, tested and validated exclusively via simulations.

Targeted applications: This dissertation is aimed specifically to open-market civil UAVs and MAVs. These benefit more from the research presented here than military UAVs, which have considerable computational power, are more often than not remotely operated by ground-crew, and of which high-fidelity models are usually developed. Additionally, while this dissertation contemplates dangerous environments for RL agents, scenarios where the UAV is intentionally threatened, as it would be the case for a military UAV, are not considered.

1.5. Related work

The working principles behind RL are extremely intuitive, and the field of application is wide, due to the generality of the approach. Because of this, many researchers have been attracted by RL, and numerous variations on the basic principle of reinforcement have been designed, with often very different research goals. The goal of this section is to introduce the reader to some of the most relevant literature and developments over the basic RL scheme, covering the three areas of safety of exploration, online learning efficiency, and robustness.

1.5.1. Safety in reinforcement learning

In the literature, a widely adopted approach to safety consists of assigning low reward for undesired transitions, such that the most reliable policy maximises the minimal sum of rewards in the presence of uncertainties and stochasticity. Safety is therefore embedded into policy performance. This *worst-case* or *minimax* [12] approach belongs to the *optimisation criterion* of safety [13]. 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 [14]. Alternatively, by assigning low reward, the variance of the return can be taken into account by adopting *risk-sensitivity* approaches [15–18].

Several techniques exist to implement both minimax and risk-sensitive methods (e.g., [19]); 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 optimisation 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. Garcia [13] refers to three different approaches: (i) "providing initial knowledge", directing the learning in its initial stage towards more profitable and safer regions of the state space [20]; (ii) "deriving a policy from demonstrations" by means of *Learning from Demonstration* [21]; and (iii) "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 [22]. 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* [23] acting as an exploration bonus towards the safer regions of the environment.

Among algorithms that directly avoid unsafe transitions, [24] 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 states. A similar approach is taken in [25], 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 [26] a variable amount of perturbation is introduced in a given safe but inefficient baseline controller, such 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, simulator or backup policy in order to prevent catastrophic exploration when facing critical decisions. Moldovan and Abbeel [27] define safety in terms of ergodicity of the exploration, and introduce an algorithm that relies on beliefs of the system, but not on a predefined baseline policy or safe controller.

Summarizing, existing safe exploration algorithms share the need for a-priori information, which derives from the knowledge of which regions should or should not be visited, from assuming specific properties of the environment, or from adopting a baseline safe policy. The methods proposed in this dissertation relax the above assumptions, allowing the agent to discover safe regions and safe policies during learning.

1.5.2. Online efficiency in reinforcement learning

Any algorithm that performs online learning faces two challenges. The main problem is that only a limited amount of training episodes can be produced when learning online. Therefore, online algorithms strive to achieve maximum efficiency of learning with a limited amount of episodes, and direct exploration toward meaningful areas of the learning space. The class of Adaptive Dynamic Programming[6] (ADP) methods [28], for example, employs Neural Networks (NN) to approximate the policy, the model, and, depending on the exact methodology applied, either the value function, its derivatives with respect to states, or both. This methodology is quite popular for solving control problems, since the more the NN approximates the true model, and value, the more efficient policy improvement cycles become.

Another technique is *fitted Q-iteration* [29, 30]. The principle behind fitted Q-iteration and similar algorithms is to exploit the data stemming from online interaction as efficiently as possible to facilitate convergence of the policy. Specifically, these methods utilize as input the tuples $\{s_k, a_k, s_{k+1}, r\}$ and submit these to regression algorithms to approximate the dependence of the instantaneous reward r from the state-action pairs (s, a). This procedure allows to adopt very sparse collections of data, which are used as a batch to obtain an estimate of the Q-value.

A different principle investigated in literature is to direct the exploration towards states and actions that are more likely to improve the policy. One notable example is *R-max* [31]. It follows the *optimism in the face of uncertainty* [32] criteria, which can be summarized as assuming that unvisited state-action pairs are optimal. R-max does so by appending a maximum reward to all such states, until a threshold number of visits, which depends on the parameters of the problem, have been produced. A method with a similar principle is *Tabu Search* [33], where exploration is guided as to avoid recently tried actions, while at the same time encouraging the agent towards promising regions of the state space.

Policy search [34–36] is an approach that exploits a policy parametrization in the form $a(s) = \pi(s; \theta)$. The fundamental idea of the approach is to search the best policy only among those obtainable with the θ parametrization. This greatly simplifies the RL problem by substituting the original learning space, consisting of all possible state-action pairs, with the reduced learning space spanned by parameters θ. Furthermore, policy search combines naturally with well known optimisation methods such as gradient descent, which makes it computationally attractive. The main obstacles to the implementation of this method are adopting a correct parametrization of π and solving the possibly nonlinear optimization problem, e.g., without getting stuck in local minima. Bagnell and Schneider [37] show an application of policy search to unmanned helicopter control.

As already mentioned, online efficiency encompasses a second aspect, which is

[6] Also referred to in literature as *Approximate* Dynamic Programming

the computational cost of modifying, invoking, and applying the agent's policy. A few attempts have been performed to reduce this cost. Degris, Pilarksi and Sutton [38] adapt previous actor-critic algorithms to continuous state and action in order to obtain an incremental method with computational cost that is linear in the number of policy parameters. Hanselmann, Noakes and Zaknich [39, 40] also investigate actor-critic methods for continuous time applications.

Nonetheless, reducing the computational cost occupies a role of secondary importance in literature when compared to accelerating the speed of learning. This can be understood considering that faster learning results in better overall performance and is therefore a prerequisite for implementation. However, as explained in Section 1.3.3, the relevance of computational costs increases when safety of exploration is considered.

1.5.3. Robustness in reinforcement learning

Robustness can be defined as the ability of a controller to perform optimally, or at least adequately, when facing uncertainties. These uncertainties can be ascribed to inaccurate reference models and/or from noise affecting the system. Model-free algorithms represent one valid option to address uncertainties, as the learning occurs without any explicit reference to a model. In that respect, the original Q-learning algorithm is sufficient to guarantee robustness and model-independence.

From a pure optimal control perspective, stability and error minimization are the main goals for increasing the robustness of a controller. Therefore, several efforts have been produced to improve the performance of model-free algorithms. As an example, Q-learning has been modified to solve the H_∞ control problem [41], which concerns the stability of the controlled system. Al-Tamimi, Lewis and Abu-Khalaf [42] solve the problem for linear discrete-time zero-sum games. Kiumarsi, Lewis and Jiang [43] improve upon this by reducing the necessary assumptions, again for linear, discrete-time systems. Luo, Wu and Huang [44] extend the solution of H_∞ to nonlinear systems. Other results for model-free methods include Yang, Liu and Wei [45], and Luo et al. [46], who develop data-based robust control for continuous time systems. Worth mentioning is also the work of Jiang and Jiang [47, 48] who extend the ADP framework into Robust Adaptive Dynamic Programming (RADP).

Arguably, a second interpretation of robustness can be found that is specific for RL, and that somewhat differs from the above, rigorous one. In the most general RL framework, a disembodied agent controls an indivisible and unspecified environment. However, in UAV and MAV applications the agent is actually embodied in a physical platform, which in turn interacts with a physical, mutable and possibly unknown "world". Moreover, this interaction is mediated by sensors and actuators that are inevitably affected by noise and errors. With respect to the more canonic uncertainty pertaining robust control, this represents a more specific source of error, deriving from the limited knowledge of the world that the agent operates within. As a result, more specialized methods to solve this inconvenience have been developed.

Partially observable Markov decision processes [32] (POMDPs) improve the learning of agents with limited information with respect to normal MDPs. Specifically, POMDPs are utilized when the agent does not know the state, but only an observation of it. The state is then a probability distribution which depends on current as well as past observations. Additionally, the environment itself, in terms of reward and transition probabilities, can also be modeled as a distribution within the POMDP framework, e.g., where some unmodeled dynamics are present. The agent must act according to its own *belief*, i.e., depending on the likelihood of being in one state (or in one world model) rather than in another, given its past observations. Numerous RL tasks have been modeled as POMDPS, from abstract games [49] to robotic path planning [50] and robotic competitions [51]. One key observation is that, in order to maximize its expected return, the agent must refine its belief through *information gathering* actions. In this regard, one class of methods worth mentioning is that of Bayesian Reinforcement Learning (see, e.g., [52–54]), which formalizes information gathering by adopting a Bayesian distribution of the beliefs.

Dynamic or non-stationary environments are a further challenge for learning even when the state is entirely observable. Both state transitions and the corresponding rewards can change over time in such an environment. RL is in itself able to cope with these changes by re-learning its value or action-value function. However, convergence might be an issue, i.e., the environment might change too radically or too fast for the agent to successfully adapt. Dynamic environments have been studied in settings as diverse as bandit problems [55], multi-agent RL [56, 57], and robot navigation [58, 59]. In addition to POMDPs, multiple strategies and algorithms have been adopted to solve these problems, such as *Experience Replay* [60], fuzzy logic [61], *Incremental Learning* [62], intrinsically motivated RL [63, 64] and Instantiated Information [65, 66]. Hierarchical Reinforcement Learning [67] (HRL) deserves a special mention. This branch of RL accelerates learning by identifying *temporally extended actions*. These are essentially "skills" or "subtasks" that the agent develops in order to sequentially solve its main task. The validity of the individual skills usually persists even as the environment changes, which mitigates the effects of learning in non-stationary environments.

Therefore, several methods exist to achieve robustness with respect to different sources of uncertainty. However, previous research has in general overlooked the implications of robustness for the problem of safety, focusing more on its benefits for the performance.

In conclusion, the novelty of the methods proposed in this dissertation, with respect to those illustrated in Section 1.5, consists of jointly addressing safety, online efficiency and robustness of the exploration process, in addition to reducing the a-priori information required from the designer in terms of dangerous states and actions.

1.6. Outline of content

When not considering Chapter 1 (this introduction) and Chapter 7, this thesis is divided in two parts. Part I, consisting of Chapters 2, 3 and 4, focuses on the three UAV RL challenges introduced in Section 1.3 and elaborates three *key approaches*, each of which addresses a specific challenge: safety, online efficiency, and robustness. Then, Part II introduces *hybrid methods*. While the methods of Part I solve the RL challenges individually, the hybrid ones of Part II are combinations of key approaches and can therefore address more challenges at the same time.

In Chapter 2, two important but marginally restrictive assumptions are made: Risk Perception and Bounding Model. It is shown how these are sufficient to introduce a new heuristic strategy for exploration that relies on the discovery of temporary safe policies, called backups. Two algorithms, SHERPA and OptiSHERPA, are presented that act as safety-filters with respect to blind search actions, resorting to backups when the agent's proposed action is potentially dangerous. The algorithms are shown to be effective but computationally demanding, which is not ideal, since in the considered case the agent must learn online.

Chapter 3 addresses the limitation of online efficiency by introducing graph structures, which more efficiently represent the uncertain bounding models, and by precomputing safety assessments directly within graphs, simplifying the search of online safe exploratory policies.

Chapter 4 addresses those cases where graph policies are not ideal due to the uncertainties originating from the unsophisticated model, and to the discretization necessary to generate a graph. Safe Reinforcement Learning is combined with the notion of hierarchy into Safe Hierarchical Reinforcement Learning (SHRL). Utilizing hierarchy is shown to yield a more robust exploration, offering efficient precomputation, higher task flexibility and reduced learning time.

Chapter 5 illustrates a hybrid approach that combines the graph formulation of Chapter 4 with the heuristic approach of Chapter 2 in the form of safety metrics. Depending on the specific metric implemented, the approach is shown to increase safety.

Chapter 6 presents an approach that integrates the risk perception and bounding model of Chapter 2, the graph representation of the dynamics of Chapter 3, and the state space hierarchical projection of Chapter 4. This unified approach is simulated in an MAV exploration task and is shown to perform optimally with respect to both safety and online efficiency under multiple operating conditions.

Chapter 7 summarizes the results and the findings of the previous chapters, and shows how this dissertation provides UAV RL agents with several methods that address the goal of safe, online, and robust exploration. Additionally, it proposes several points of further development for the machine learning and the UAV operator communities.

1.7. List of publications

This section lists the publication sources for each chapter:

- Chapter 2 is based on the following publication:

  T. Mannucci, E. van Kampen, C. C. de Visser and Q. Chu, ”Safe Exploration Algorithms for Reinforcement Learning Controllers”, in IEEE Transactions on Neural Networks and Learning Systems, accepted and awaiting publication.

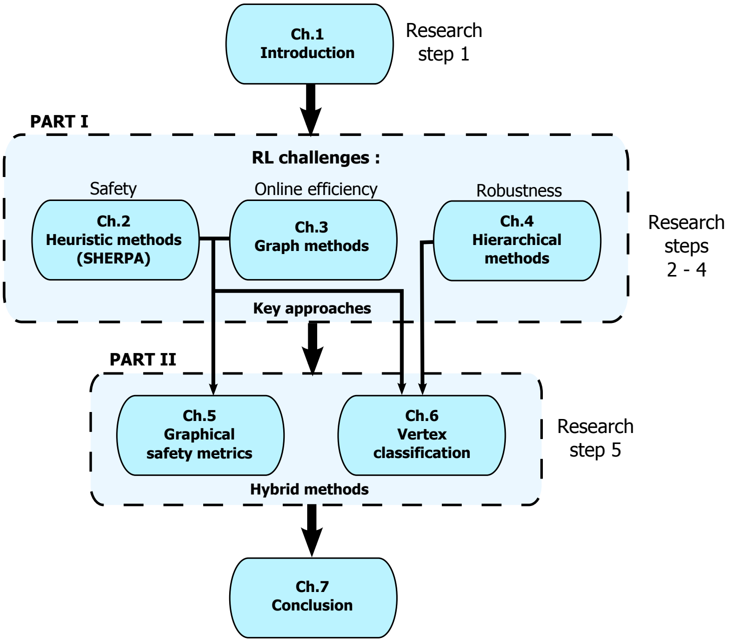


Figure 1.4: Structure of the chapters in the dissertation.

- Chapter 3 is based on the following paper:

  T. Mannucci, E. van Kampen, C. C. de Visser, and Q.Chu. ”Graph based dynamic policy for UAV navigation”, in Proceedings of the AIAA Guidance, Navigation, and Control Conference, AIAA SciTech Forum 2016, San Diego, CA.

- Chapter 4 is based on the following papers:

  T. Mannucci and E. van Kampen, ”A hierarchical maze navigation algorithm with Reinforcement Learning and mapping,” 2016 IEEE Symposium Series on Computational Intelligence (SSCI), Athens, Greece, 2016, pp. 1-8;

  T. Mannucci, E. van Kampen, C. C. de Visser, and Q. Chu, ”Hierarchically Structured Controllers for Safe UAV Reinforcement Learning Applications”, in Proceedings of the AIAA Information Systems-AIAA Infotech, AIAA SciTech Forum 2017, Grapevine, TX.

- Chapter 5 is based on the following paper:

  T. Mannucci, E. van Kampen, C. C. de Visser, and Q. Chu. ”A novel approach with safety metrics for real-time exploration of uncertain environments”, in Proceedings of the AIAA Guidance, Navigation, and Control Conference, AIAA SciTech Forum 2016, San Diego, CA.

- Chapter 6 is based on the following paper:

T. Mannucci, E. van Kampen, C. C. de Visser, and Q. Chu, ”Safe and Autonomous UAV Navigation using Graph Policies”, in Proceedings of the AIAA Information Systems-AIAA Infotech, AIAA SciTech Forum 2017, Grapevine, TX.

I

KEY APPROACHES

2
Heuristic methods

As demonstrated in Chapter 1, safety is a primary concern when performing online exploration for UAV Reinforcement Learning agents. A preliminary step to develop a new and effective safe exploration strategy to answer the ***challenge of safety*** *would be to formalize the risk posed by the environment for the agent. Chapter 2 serves therefore two purposes. The first is to develop a mathematical representation of danger in the learning environment, and to integrate this within the Markov decision process framework. The second purpose is to formulate an exploration strategy that, given the integrated framework, can prevent dangers to the agent without resorting to a-priori known safe policies.*

This chapter is a copy of *T. Mannucci, E. van Kampen, C. C. de Visser and Q. Chu*, "Safe Exploration Algorithms for Reinforcement Learning Controllers", in IEEE Transactions on Neural Networks and Learning Systems, accepted and awaiting publication

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 work 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.

2.1. 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[68] 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 dependency[69, 70]. A model-free option amidst adaptive control is Reinforcement Learning (RL).

Reinforcement Learning is a knowledge based control scheme that mimics animal development[71]. 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 Reinforcement Learning: stimuli constitute the *plant* or *system*; the animal is the *agent* following a temporary behavior, i.e. a *policy*. Lastly, 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 (PAC)[72], and proofs of near-optimality and optimality for both discrete and continuous[73] applications were found. RL has also proved its worth in combination with Neural Networks (NN) in the development of Neuro-Dynamic Programming[74, 75], where the approximation power of NNs is utilized to efficiently represent both the value and the policy, thus reducing the "curse of dimensionality"[76]. In recent years, multiple successful applications[77–80], utilize RL as a framework for learning, in particular *model-free* RL[81–83].

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 sub-optimal 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 an hostile environment where the consequences of wrong actions are not limited to "bad" performance, but include long term effects that can't 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 work is to avoid such occurrences during learning, thus achieving *safe exploration*[84].

In the literature, a widely adopted approach to safety consists of assigning negative reward for undesired transitions, such that the most reliable policy maximises the minimal sum of reward in the presence of uncertainties and stochasticity. Safety is therefore embedded into policy performance. This *worst-case* or *minimax* [12] approach belongs to the *optimisation criterion* of safety [13]. 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 [14]. Alternatively, by assigning negative reward, the variance of the return can be taken into account by adopting *risk-sensitivity* approaches [15–18]. Several techniques exist to implement both minimax and risk-sensitive methods (see e.g. [19]); 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 optimisation 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. Garcia[13] differentiates three different approaches: "providing initial knowledge", directing the learning in its initial stage towards more profitable and safer regions of the state space [20]; "deriving a policy from demonstrations" by means of *Learning from Demonstration*

[21]; "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[22]. 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* [23] acting as an exploration bonus towards the safer regions of the environment.

Among algorithms that directly avoid unsafe transitions, [24] 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 [25], 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 [26] 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[27] 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"[32] 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.

2.2. 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.

2.2.1. 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* u that the agent can select. $\mathcal{D}$ is a mapping $\mathcal{S} \times \mathcal{A} \times \mathcal{S} \rightarrow [0,1]$ that $\forall \boldsymbol{x}$, $\boldsymbol{x}' \in \mathcal{S}$ and $\forall u \in \mathcal{A}$ assigns to triplet $(\boldsymbol{x}, u, \boldsymbol{x}')$ the probability of the environment transitioning from state $\boldsymbol{x}$ to state $\boldsymbol{x}'$ given action u. Function $\mathcal{R} : \mathcal{S} \times \mathcal{S} \rightarrow \mathbb{R}$ 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 $\pi : \mathcal{S} \times \mathcal{A} \rightarrow [0,1]$ is a mapping that $\forall \boldsymbol{x}$ assigns a probability of selecting action 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 [85] 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 [73].

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.

2.2.2. 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 $\tau = (\boldsymbol{x}, u, \boldsymbol{x}')$, $\boldsymbol{x}, \boldsymbol{x}' \in \mathcal{S}$, $u \in \mathcal{A}$. Let $\mathcal{T}_{\text{fat}}$ represent the subset of those transitions generating fatal occurrences. The following will be assumed:

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

This assumption is not new to the literature (e.g., see [86]) 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*) $\mathrm{FSS} = \{\boldsymbol{x}' | \forall \boldsymbol{x} \in \mathcal{S},\ \forall u \in \mathcal{A},\ \tau = (\boldsymbol{x}, u', \boldsymbol{x}') \in \mathcal{T}_{\text{fat}}\}$

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

Definition 2 (*Safe state space*) $\mathrm{SSS} = \mathcal{S} \setminus \mathrm{FSS}$.

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:

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 Figure 2.1). 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* k^{th} *risk,* $k \in \{1, \cdots, m\}$*. Let* $\boldsymbol{x}_k$ *be the projection of state* $\boldsymbol{x} \in \mathcal{S}$ *to* RSS_k*, and* $\boldsymbol{f}_k \in \mathrm{RFSS}_k$ *the nearest fatal state to* $\boldsymbol{x}_k$*. Then* $\forall k \in \{1, \cdots, m\}$*,* $\exists \epsilon_k \in \mathbb{R}$ *such that at each time-step, the value of the boolean function*

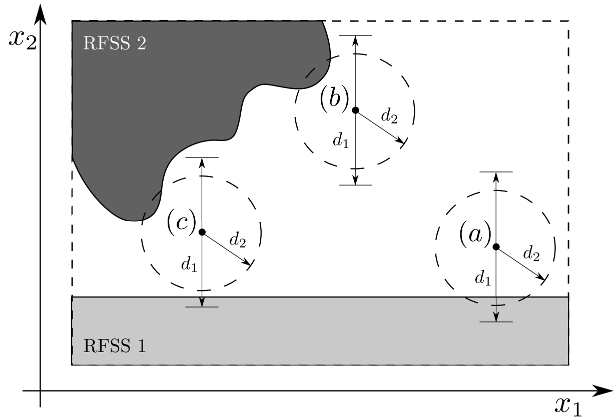


Figure 2.1: An agent exploring a state space (bold dotted rectangle) with two different RFSS. The first cause of fatality is independent from the value of component x_1, and an Euclidean distance d_1 is provided for the risk perception. A second cause of fatality is dependent on both components of the state so that an Euclidean distance d_2 in the whole state space is given. In (a), RFSS 1 is in reach; in (b), RFSS 2 is in reach; in (c), both RFSSs are in reach. In all three cases the agent receives an identical warning.

$$\|\boldsymbol{x}_1 - \boldsymbol{f}_1\| \leq \epsilon_1 \lor \cdots \lor \|\boldsymbol{x}_m - \boldsymbol{f}_m\| \leq \epsilon_m \tag{2.1}$$

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

The risk perception can be represented by a unknown mapping $W : \mathcal{S} \to \{0,1\}$, $W(\boldsymbol{x})$ equal to 1 if Eq. (2.1) 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 $\boldsymbol{\sigma}(\boldsymbol{x}(t_0), u(t), t)$ the state of the system at time $t > t_0$ when applying generic action history $u(\boldsymbol{x}(t_0), t)$:

$$\boldsymbol{\sigma}(\boldsymbol{x}(t_0), u(t), t) = \boldsymbol{x}(t_0) + \int_{t_0}^{t} \dot{\boldsymbol{x}}(\boldsymbol{x}, u(\tau))d\tau, \tag{2.2}$$

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

Definition 3 (*Lead-to-fatal states*)

$$L = \{\boldsymbol{x} \,|\forall u(x(t_0), t)\,,\, \exists t : \boldsymbol{\sigma}(x(t_0), u(t), t) \in \text{FSS}\}. \tag{2.3}$$

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 [87], and as the *supercritical* states defined in [24] for a deterministic MDP.

In model-based RL, the controller has an adaptive model of the environment that allows off-line training and on-line system identification. Instead, in model-free RL (e.g., Q-learning [6]) there is no off-line training - the policy refinement is based on on-line training and no model is generated. An intermediate position between this two instances will be adopted in this work by introducing the following definition:

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

$$\forall \tau = \{\boldsymbol{x}, u, \boldsymbol{x}'\} \in \mathcal{T};\ \boldsymbol{x}' \in \Delta(\boldsymbol{x}, u) \tag{2.4}$$

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.

2.3. 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[88]. 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.

2.3.1. Interval analysis

An interval $[\boldsymbol{x}] = [\underline{\boldsymbol{x}}, \overline{\boldsymbol{x}}]$ is the set $\{\boldsymbol{x} | \underline{x}_i \le x_i \le \overline{x}_i, i \le n\}$, $\underline{\boldsymbol{x}} = (\underline{x}_1, \cdots, \underline{x}_n)^T$, $\overline{\boldsymbol{x}} = (\overline{x}_1, \cdots, \overline{x}_n)^T$, $\underline{x}_i$, $\overline{x}_i \in \mathbb{R}$. 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 $\mathcal{D}(\boldsymbol{x}, u, \rho_1, \ldots, \rho_k)$, with parameters $\rho_i \in [\rho_i]$ and $[\rho_i]$ known confidence intervals. Following the above definition, $[\mathcal{D}(\boldsymbol{x}, u)] = \{\mathcal{D}(\boldsymbol{x}, u, \rho_1, \ldots, \rho_k) | \rho_i \in [\rho_i]\}$ is the bounding model of $\mathcal{D}$. For each $\boldsymbol{x}$ and u, $[\mathcal{D}(\boldsymbol{x}, u)]$ 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 *dependency problem*, which propagates uncertainty in states and parameters.

2.3.2. Background and algorithm description

The final goal of the agent is to select at each time t a control 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 $u(t)$, $t \in [t_1\ t_2]$ is **safe** in state $\boldsymbol{x}(t_1)$ if*

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

Safe control $u(t)$ is *feasible* if $\forall t$, $\underline{u} \leq u(t) \leq \overline{u}$, where $\underline{u}$ and $\overline{u}$ represent boundaries on control, e.g., saturation. If this condition is strictly satisfied, $u(t)$ 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 conditon of Eq. (2.3) 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 $u(t)$ with associated trajectory $\boldsymbol{\sigma}(\boldsymbol{x}(t_0), u, t)$, $t \in [t_0, t_1]$ if

$$\exists t', t'' \in [t_0, t_1], t' \neq t'' \colon \boldsymbol{\sigma}(\boldsymbol{x}(t_0), \mathbf{u}, t') = \boldsymbol{\sigma}_u(\boldsymbol{x}(t_0), \mathbf{u}, t'') \tag{2.5}$$

then there exists a safe and feasible control $u^*(t)$ for $t \in [t_0, \infty)$. It can be seen from Eq. (2.2) that Eq. (2.5) can be rewritten as $\boldsymbol{\sigma}(\boldsymbol{x}(t'), u(t-(t'-t_0)), t') = \boldsymbol{\sigma}(\boldsymbol{x}(t'), u(t-(t'-t_0)), t'')$. Then, $\forall \Delta t \leq (t''-t')$, $\boldsymbol{\sigma}(\boldsymbol{x}(t''), u(t-(t''-t')-(t'-t_0)), t''+\Delta t) = \boldsymbol{\sigma}(\boldsymbol{x}(t'), u(t-(t'-t_0)), t'+\Delta t)$. Since trajectory $\boldsymbol{\sigma}(\boldsymbol{x}(t'), u(t), t'+\Delta t) \in \mathrm{SSS}$, 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 Eq. (2.5) 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 $u_b(t)$, $t \in [t_0, \bar{t}\,]$ is an* ***ideal backup*** *from $\boldsymbol{x}_0$ to $\boldsymbol{x}(\bar{t})$ if*

$$\exists\ t < \bar{t} : \boldsymbol{x}(\bar{t}) = \boldsymbol{x}(t). \tag{2.6}$$

An ideal backup guarantees safety; however, Eq. (2.6) 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 $u(t)$ 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 $u^*(t)$ a safe control for $\boldsymbol{x}$: $\|\boldsymbol{x} - \boldsymbol{x}_0\| < \epsilon$ and $t \in [t_0, \bar{t}]$ such that*

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

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 $u_b(t)$, $t \in [t_0, \bar{t}]$ is a* ***backup*** *from $\boldsymbol{x}_0$ to $\boldsymbol{x}(\bar{t})$ with reach ϵ if*

$$\exists\ t < \bar{t} : \|\boldsymbol{x}(\bar{t}) - \boldsymbol{x}(t)\| \leq \epsilon \tag{2.7}$$

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

which in turn allows for the following theorem:

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

$$u(t) = \begin{cases} u_b(t) & \textit{if } t \leq t_2 \\ u^*(t) & \textit{otherwise} \end{cases}$$

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

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

Theorem 1 is of limited practical use - since it relies on the existence of $u_I(t)$ and does not specify the actual value of ϵ - but formalizes the observation contained in Eq. (2.5) in the presence of uncertainty. The equilibrium condition can be formalized as follows:

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

The definition of backup can be revised as

Definition 9 *A feasible, safe control action $u_b(t)$, $t \in [t_0\ \bar{t}]$ 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. $\|\boldsymbol{x}(t_2) - \boldsymbol{x}_E\| \leq \delta$.*

2.3.3. 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 proceding further, it is convenient to slightly alter the definition of backup to accommodate the use of intervals. Let the bounding model Δ of $\mathcal{D}$ be its IA extension $[\mathcal{D}]$ with time-discrete formulation, so that $[\boldsymbol{x}_{k+1}] = \Delta([\boldsymbol{x}_k], u_k)$, and let $[\epsilon]$ and $[\delta]$ be two n-dimensional symmetric intervals. Then, a backup can be reformulated as follows:

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

$$\forall i \in \{1, 2, \cdots, m\},\ [\boldsymbol{x}_{k+i}] \subset \mathrm{SSS} \tag{2.8}$$

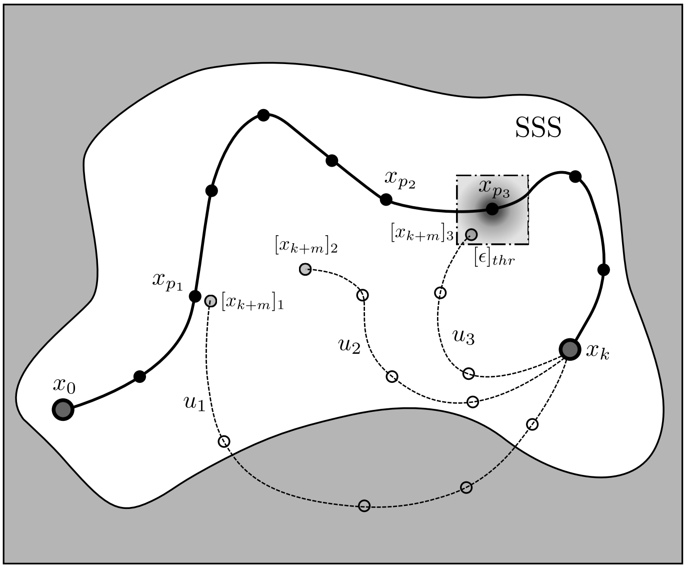


Figure 2.2: An example of backup evaluation. Starting from position $\boldsymbol{x}(0) = \boldsymbol{x}(t_0)$, the agent has followed its trajectory to current state $\boldsymbol{x}_k$. The three dashed lines represent the means of three proposed interval backups from $\boldsymbol{x}_k$. The proposed backup generated by control u_1 is discarded since it violates the known safe state space SSS. The proposed backup generated by 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 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.

and :

$$\forall \boldsymbol{x}_{k+m} \in [\boldsymbol{x}_{k+m}], \exists p \leq k \ : (\boldsymbol{x}_{k+m} - \boldsymbol{x}_p) \in [\epsilon] \tag{2.9}$$

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

$$\forall \boldsymbol{x}_{k+m} \in [\boldsymbol{x}_{k+m}], (\boldsymbol{x}_{k+m} - \boldsymbol{x}_E) \in [\delta] \tag{2.10}$$

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

This definition replaces the reach with intervals $[\epsilon]$ and $[\delta]$. Given two symmetric threshold intervals $[\epsilon]_{thr}$ and $[\delta]_{thr}$, a backup is effective if either $[\epsilon] \subset [\epsilon]_{thr}$ or $[\delta] \subset [\delta]_{thr}$. Figure 2.2 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 $[\epsilon]_{thr}$ should depend on the magnitude of the control. Given Assumption 4, $\lim_{\epsilon \to 0} \max_{k} \|u(t_k) - u^*(t_k)\| = 0$, so that the smaller the threshold $[\epsilon]_{thr}$, the more likely the feasibility of control $u(t_k)$. To account for this, an initial threshold interval $[\epsilon]_{max}$ is assigned to each new state $\boldsymbol{x}_k$. At each successive time-step, all the threshold intervals are shrunk by a factor η:

$$\eta = (\prod_{i=1}^{m} \eta_i)^{\frac{1}{m}}\ ;\ \eta_i = \left(1 - \frac{|u_i(t_k) - \frac{1}{2}\cdot(\underline{u}_i + \overline{u}_i)|}{\frac{1}{2}\cdot(\overline{u}_i - \underline{u}_i)}\right)^{\lambda} \tag{2.11}$$

with positive exponent $\lambda < 1$. The more the control action nears the boundaries $\underline{u}$ and $\overline{u}$, the smaller the η, and the more the shrinkage of $[\epsilon]_{thr}$. 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 $[\epsilon]_{max}$ are closely related; e.g., a small λ should be paired with a small $[\epsilon]_{max}$ and a high λ with an increased $[\epsilon]_{max}$.

2.3.4. SHERPA

In this section, the SHERPA algorithm (Figure 2.3) 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 u_p generated by policy π, SHERPA computes the predicted interval $[\boldsymbol{x}_p] = \Delta(\boldsymbol{x}_0, u_p)$. If $[\boldsymbol{x}_p] \notin \mathrm{SSS}(t_0)$, 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 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 the 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[89] were successfully adapted for robotic navigation and path planning with lack of environmental knowledge, e.g uncertainty in observed obstacles[90], in the number of obstacles[91] and even in the entire environment[92]. 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[93], Control Barrier Functions[94] and Control Lyapunov-Barrier Functions[95] 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.

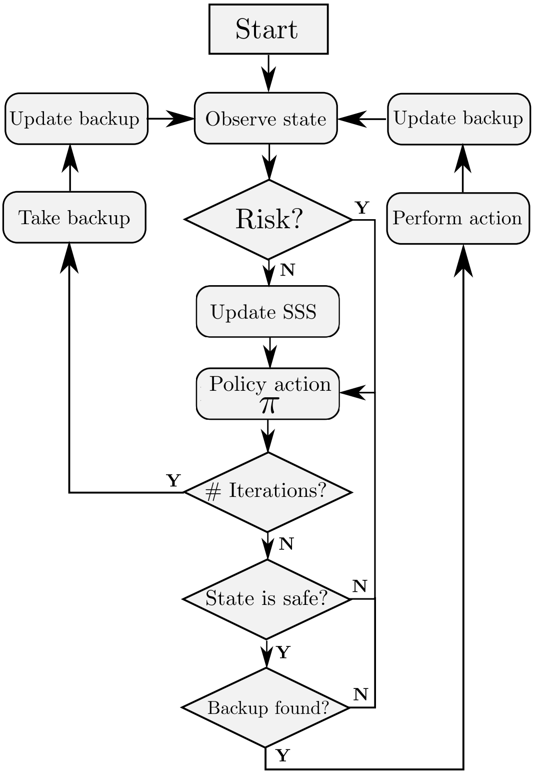


Figure 2.3: Flowchart summarizing the procedure of SHERPA. The policy action is checked for safety and for backups; if the checks succeed, the action is taken and the new backup is stored in place of the previous. If the iteration limit is reached without success, SHERPA recollects the previously stored backup.

2.3.5. Quadrotor task

In this section, a task is simulated for a two-dimensional 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 $(x,\ \dot{x},\ z,\ \dot{z},\ \theta)^T$, where x, z and θ are the horizontal and vertical coordinate of the quadrotor, and the pitch angle. The bounding model[2] is

$$\ddot{x} = -\frac{[c]}{[m]}|\sin\theta|\,|\dot{x}|\,\dot{x} + \frac{T}{[m]}\sin\theta\ ;\ \dot{\theta} = [k]I_\tau$$

$$\ddot{z} = -\frac{[c]}{[m]}|\cos\theta|\,|\dot{z}|\,\dot{z} + \frac{T}{[m]}\cos\theta - g \qquad (2.12)$$

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

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

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.

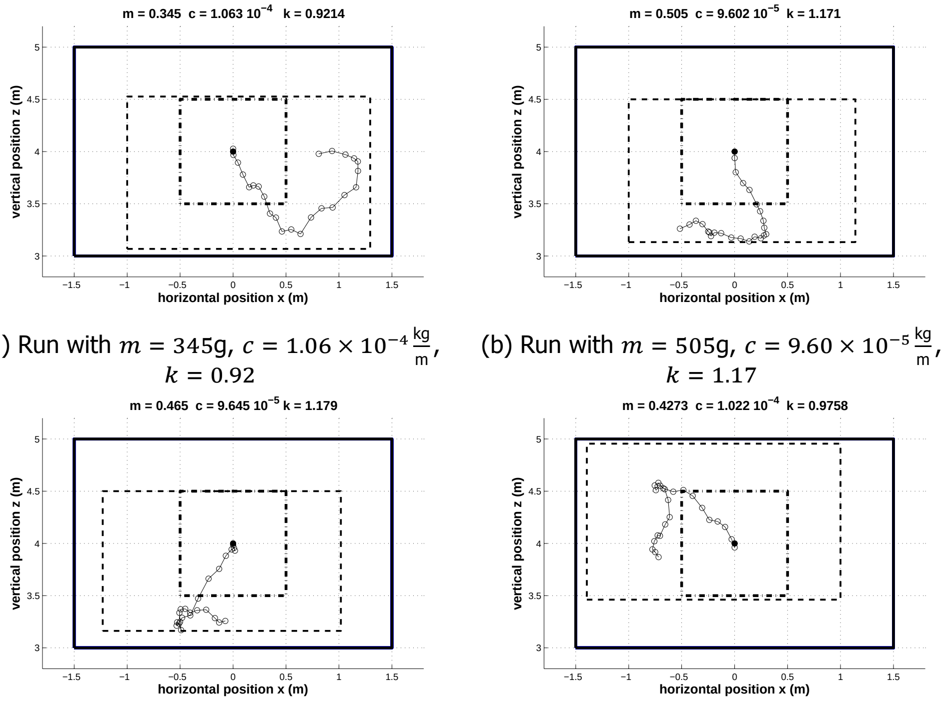
The quadrotor initially hovers at the middle of a room three meters wide and two meters high. Colliding with the walls or with the floor/ceiling is a fatal occurrence, and the quadrotor perceives risk half a meter from the ceiling or floor, or at one meter from a wall. The reach of the backup depends on $[\epsilon^{max}] = ([-\frac{1}{2}\,,\ \frac{1}{2}]\text{m};\ [-\frac{1}{2}\,,\ \frac{1}{2}]\frac{\text{m}}{\text{s}};\ [-\frac{1}{2}\,,\ \frac{1}{2}]\text{m};\ [-\frac{1}{2}\,,\ \frac{1}{2}]\frac{\text{m}}{\text{s}};\ [-\frac{\pi}{6}\,,\ \frac{\pi}{6}]\,)$. 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 Eq. (2.11) to vanish. Coefficient λ is chosen as 0.5. As for $[\delta]$, taking into consideration the dynamics in Eq. (2.12), it is evident that the position (x, y) of the quadrotor itself does not influence the value of the derivatives. Therefore the interval $[\delta_{thr}] = ([-\infty, \infty];\ [-1, 1]\frac{\text{m}}{\text{s}};\ [-\infty, \infty];\ [-1, 1]\frac{\text{m}}{\text{s}};\ [-45, 45]°)$ with equilibrium point $\dot{x} = 0$, $\dot{z} = 0$, $\theta = 0$ is selected. SHERPA evaluates up to 10 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, the $SSS(t_0)$ coincides with

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

and it is updated whenever $W = 0$ as

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

[2] It can be shown that, due to local Lipschitz continuity of Eq. (2.12)), Assumption 4 is valid for any realization of the dynamics.



(a) Run with $m = 345\text{g}$, $c = 1.06 \times 10^{-4}\frac{\text{kg}}{\text{m}}$, $k = 0.92$

(b) Run with $m = 505\text{g}$, $c = 9.60 \times 10^{-5}\frac{\text{kg}}{\text{m}}$, $k = 1.17$

(c) Run with $m = 465\text{g}$, $c = 9.64 \times 10^{-5}\frac{\text{kg}}{\text{m}}$, $k = 1.18$

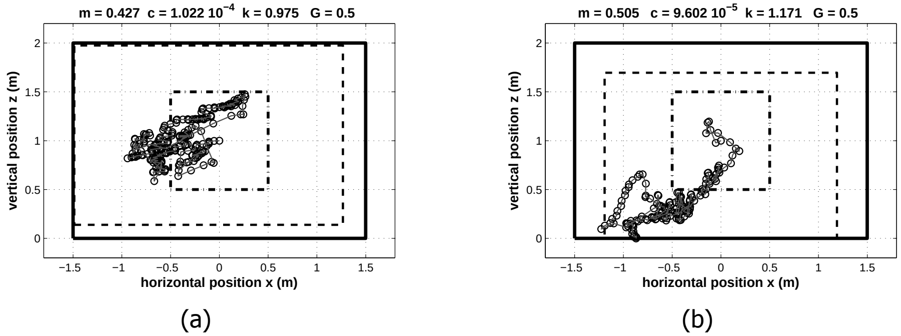
(d) Run with $m = 427g$, $c = 1.02 \times 10^{-4}\frac{\text{kg}}{\text{m}}$, $k = 0.98$

Figure 2.4: The dots represent the quadrotor position in time. The solid rectangle represents the fatal states that the agent must avoid. The dot-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 series of four runs of SHERPA is shown in Figure 2.4. 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 three out of four cases, the quadrotor reaches the boundaries of the SSS, but manages to safely divert its trajectory (Figure 2.4a and Figure 2.4b) or to wait in position until another path has been found (Figure 2.4c).

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 non exploratory, goal-oriented policy.

In order to highlight the novelty of the proposed approach, Figure 2.5 shows an application of a different method, specifically the Basic Potential Field (BPF)



(a)

(b)

Figure 2.5: The trajectory obtained when using the BPF to control a quadrotor. In a), with mass and torque efficiency in almost nominal conditions, the trajectory is centered around the middle of the known $\mathcal{SSS}$. In b), with increased mass and torque efficiency, the quadrotor flies predominantly in the lower half of the room before colliding with the floor.

method[96] for collision avoidance. The BPF differs from other potential field methods such as [89] 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 $\mathcal{SSS}$. 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 Figure 2.5a is in an almost nominal condition of mass and torque effectiveness (as in Figure 2.4d). It can be seen how a BPF with gain $G = 0.5$ prevents collisions. The trajectory is centered around the middle of the known $\mathcal{SSS}$, 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 $\mathcal{SSS}$ is not penalized with respect to its interior (see, e.g., Figure 2.4a). Figure 2.5b shows the resulting trajectory when applying the previous BPF to a quadrotor with increased mass and torque effectiveness, as in Figure 2.4b. 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 $\mathcal{SSS}$, 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 two different conditions of Figure 2.5a and Figure 2.5b. 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.

2.4. OptiSHERPA

2.4.1. Motivation and algorithm description

The previous section 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 on-line application by allowing OptiSHERPA to rank its options and to take an informed decision if the available amount of on-line 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.

2.4.2. Metrics

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,

$$d(\boldsymbol{x}, I) = \left\| (\boldsymbol{x} - \frac{\overline{I} + \underline{I}}{2}) \odot \boldsymbol{v}_{\mathrm{r}} \right\| + \rho \cdot \left\| (\frac{\overline{I} - \underline{I}}{2}) \odot \boldsymbol{v}_{\mathrm{r}} \right\| \tag{2.13}$$

is a "distance"[3] between $\boldsymbol{x}$ and the center of the interval, rescaled by $\boldsymbol{v}_{\mathrm{r}} \in \mathbb{R}^{n+}$, 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. Figure 2.6 shows this transition. Vector $\boldsymbol{v}_{\mathrm{r}}$ should be chosen so that, for any two excursions in state $\Delta\boldsymbol{x}^{\mathrm{I}}$ and $\Delta\boldsymbol{x}^{\mathrm{II}}$, it is $\|\Delta\boldsymbol{x}^{\mathrm{I}} \odot \boldsymbol{v}_{\mathrm{r}}\| > \|\Delta\boldsymbol{x}^{\mathrm{II}} \odot \boldsymbol{v}_{\mathrm{r}}\|$ iff 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 $\boldsymbol{v}_{\mathrm{r}}$ 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 a similar method as the one illustrated in Eq. (2.11). At each time-step compute η so that the metric distance between $\boldsymbol{x}$ and interval $I(t_i)$ at time at time t_j is:

$$d_{\eta}(\boldsymbol{x}, I(t_i), t_j) = d(\boldsymbol{x}, I(t_i)) / \prod_{k=i}^{j} \eta(t_k) \tag{2.14}$$

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.

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,

[3] Eq. (2.13) 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 Eq. (2.13) is restricted to the subset of "crisp" intervals, it is analogue to the Euclidean distance in $\mathbb{R}^n$, hence the denomination.

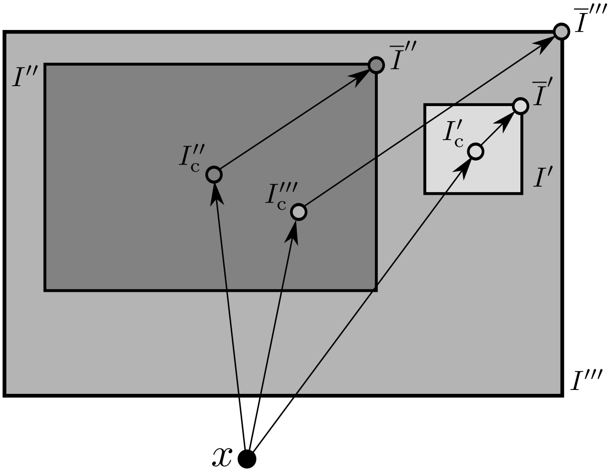


Figure 2.6: The choice of ρ affects the outcome of the metric when comparing different intervals. For lower values of ρ, interval I''' is the nearest since its center I'''_{c} is the nearest to x. By gradually increasing ρ, 2.13 will select interval I''. Finally, for ρ approaching 1, I' will be the nearest.

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 τ those τ that intersect a fatal region entirely;
4. remove from τ those τ that do not end up in the SSS, unless this depletes τ;
5. for each $\tau_j \in \tau$ compute the volume of the intersection ρ_{ij} with region R_i;
6. the optimal control sequence u^* is the one generating $\tau^* = \underset{\tau_j \in \tau}{\operatorname{argmin}} \sum_{R_i} \rho_{ij} \cdot w_i$;
7. apply the first element of u^*;
8. if $W = 1$, go to 1; otherwise exit.

Figure 2.7 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.

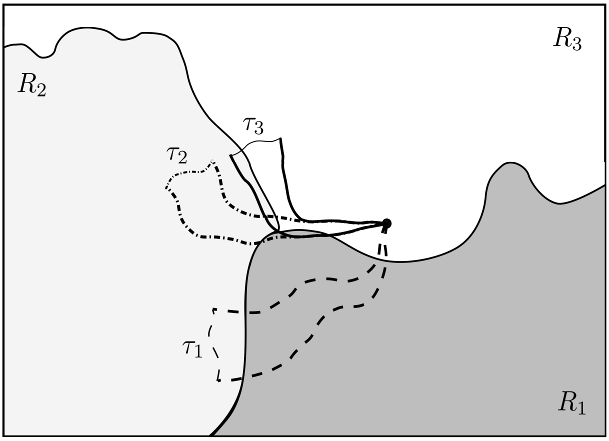


Figure 2.7: $\mathcal{S}$ comprises fatal region R_1, unknown region R_2, and safe region R_3. All τ have a non-empty intersection with R_1. However, τ_1 crosses R_1 in its entirety and is therefore the least safe; τ_2 and τ_3 also cross the region, but τ_3 has the minimal overlap with both R_1 and R_2, and the biggest overlap with R_3: it is therefore the safest.

2.4.3. OptiSHERPA

Figure 2.8 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 2.4.2, an optimal control sequence u^* is found, and its first action is implemented. The evasion metric is used until $W(\boldsymbol{x}(t)) = 0$, after which the distance metric is reimplemented. This procedure is followed until the end of exploration.

2.4.4. Elevator control task

Model

The goal of the application is to simulate on-line training for a RL agent on-board of a fighter aircraft, which is constrained to fly in a range of $\pm 40\mathrm{ft}$ from starting height. It is assumed that a stall occurs if the angle of attack $\alpha < -15°$ or $\alpha > 12°$. Violating this envelope causes a fatal occurrence in this environment. A bounding model of the aircraft is available as in Eq. (2.15) and Eq. (2.16):

$$\dot{h} = V \cdot \sin(\theta - \alpha) \tag{2.15}$$

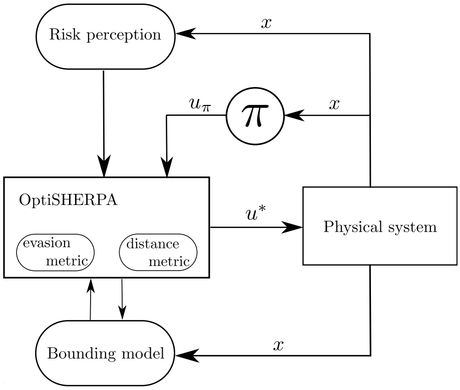


Figure 2.8: The architecture of OptiSHERPA. At each time-step, the algorithm receives information in the form of current risk perception and suggested policy control u_π. The bounding model is used to predicted trajectories, which in turn are evaluated by implementing either the distance or the evasion metric depending on current risk perception. Then the first action of u^* is taken, which generates a new state.

$$\begin{pmatrix} \dot{\theta} \\ \dot{\alpha} \\ \dot{q} \end{pmatrix} = \begin{bmatrix} 0 & 0 & 1 \\ 0 & c_{\alpha\alpha} & c_{\alpha q} \\ 0 & c_{q\alpha} & c_{qq} \end{bmatrix} \cdot \begin{pmatrix} \theta \\ \alpha \\ q \end{pmatrix} + \begin{bmatrix} 0 \\ c_{\alpha\delta_e} \\ c_{q\delta_e} \end{bmatrix} \cdot (\delta_e + \Delta\delta_e) + \begin{pmatrix} 0 \\ 0 \\ \Delta\dot{q} \end{pmatrix} \tag{2.16}$$

with

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

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

The intervals of Eq. (2.15) and Eq. (2.16) are obtained as follows. First, the nonlinear dynamics are linearised at 15000 ft of height and $500\frac{\text{ft}}{\text{s}}$ 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 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:

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

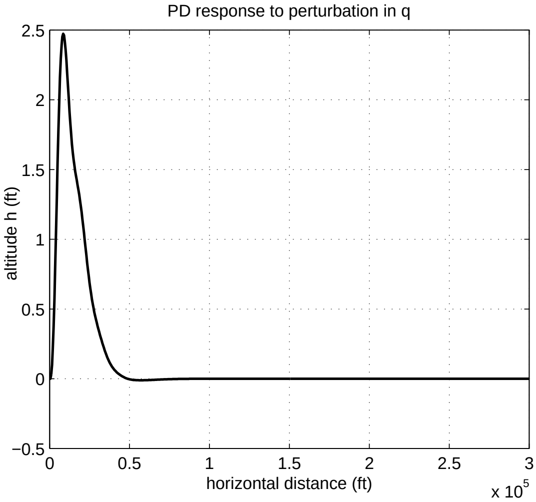


Figure 2.9: The change in flight height of the system equipped with the PD controller after a perturbation in q. The final error in height is brought to zero.

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:

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

The controller is hand tuned to achieve a satisfactory performance in nominal conditions. Figure 2.9 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 $\delta = \delta_{\mathrm{PD}}$ 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 Eq. (2.15). 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 $15\mathrm{ft}$ 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. The 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 $0.2\mathrm{s}$. Then, if danger is perceived, or if $|\gamma| > 0.5^\circ$, the agent enables OptiSHERPA. When enabled, the algorithm generates multiple command sequences, each lasting six time-steps, in the form $u = (u_0, u_b, \ldots, u_b)$, $u_0, u_b \in \{-0.75^\circ, -0.25^\circ, 0^\circ, 0.25^\circ, 0.75^\circ\}$. Command u_0 is the OptiSHERPA proposed action, and the constant u_b acts as a backup. This formulation of u reduces the total number of metric evaluations from 5^6 to 25. The distance Eq. (2.14) is computed for all the above sequences, with $\boldsymbol{v}_\mathrm{r} = [0.2(\mathrm{ft})^{-1}, \frac{36}{\pi}, \frac{36}{\pi}, \frac{4}{\pi}\mathrm{s}]^T$, $\rho = 0.5$, $\lambda = 0.1$.

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:

- a "black" region R_bk comprising all states with fatal values of h (or α);
- a "white" region R_wh comprising all states within risk perception of the nearest fatal value;
- a "green" region R_gn equal to the $\mathrm{SSS} \setminus R_\mathrm{wh}$;
- a "grey" 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

$$w_\mathrm{bk} = 2,\ w_\mathrm{r} = 0,\ w_\mathrm{gy} = -0.1,\ w_\mathrm{w} = -0.2,\ w_\mathrm{bl} = -1,\ w_\mathrm{gn} = -2.$$

A total of 500 trials is performed from the starting perturbed condition $h = \alpha = \theta = 0$, $q = 0.5^\circ$. Each trial has a duration of 600 timesteps, equivalent to $120\mathrm{s}$. 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.

Figure 2.10 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 in feet allowed by the risk perception. Figure 2.11 shows one such case. Prior to the violation, a series of positive random deflections reduce 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 Figure 2.11). 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.

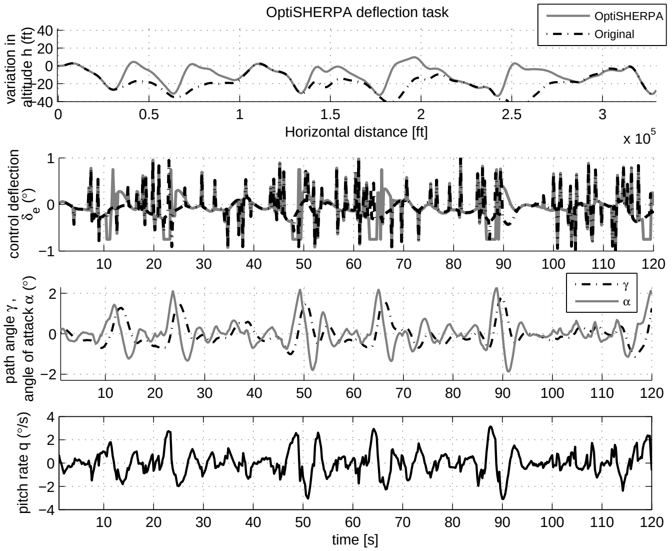


Figure 2.10: The top plot represents the trajectory with the original RL agent and with the addition of OptiSHERPA. The 2^{nd} plot represents deflection δ_e for the two different agents; OptiSHERPA selects constant maximal deflections during recovery by optimising the evasion metric. The 3^{rd} shows the variations of γ and α during flight with OptiSHERPA. The bottom plot shows pitch rate q.

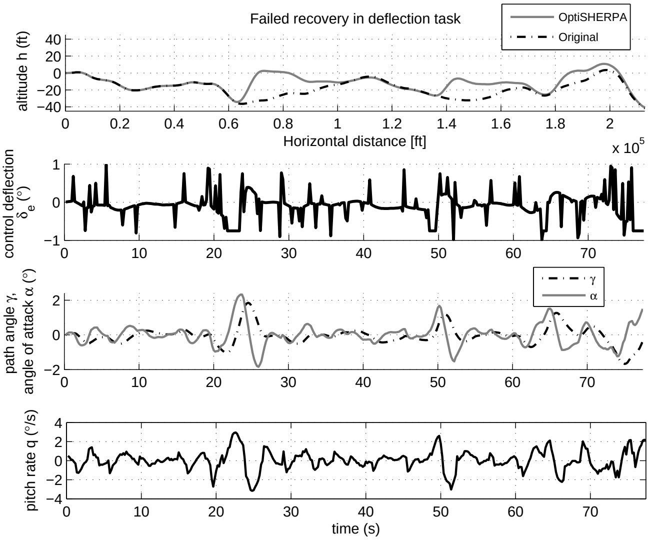


Figure 2.11: A typical violation of the flight height constraints.

2.5. Conclusions

This work presents a new approach for autonomous agents in dangerous environments. The presence of fatal and lead-to-fatal 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 Reinforcement Learning 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.

3

Graph methods

*This chapter introduces graphs as an efficient representation of the uncertain dynamics of the environment bounding model, presented in Chapter 2. The advantage of replacing the model with an uncertain graph lies in reducing the complexity required for computations, such as the backup search of Chapter 2. This addresses the **challenge of online efficiency**. Additionally, this chapter introduces **graph pruning**, a method to perform safety assessments of policies, given that a safe subset of the state space is available in the form of an **operational envelope**.*

3.1. Introduction

Chapter 2 described a heuristic approach with the main goal of providing safety during exploration. Two different algorithms based on this approach were developed with the objective of online exploration. The two algorithms select actions and propagate a continuous state, continuous time bounding model in time to search for backups.

However, numerous attempts might be necessary before a backup is found, which is computationally demanding and challenges the *online efficiency* of the agent. Chapter 3 aims to reduce the computational burden of reachability predictions, specifically by investigating the use of graphs for the representation of the environment dynamics. Due to their simplicity, these representations can be used to reduce the amount of computations required.

A further approach employing graph representations, *graph pruning*, is introduced in this chapter. This method enables the agent to check if its current policy is feasible, i.e., if it guarantees respecting the operational envelope (OE) of the task.

Section 3.2 introduces graphs, the concept of operational envelopes, and explains how to efficiently generate graphs online. Section 3.3 shows the graph pruning method together with a simulated application to an MAV navigation task. Section 3.4 concludes the chapter.

3.2. Graph representation of the environment dynamics

As introduced in Chapters 1 and 2, UAV RL agents necessitate of algorithms that, in addition to being safe, are computationally simple and can be performed online. This section presents a method to generate a graph representation of the system dynamics that can be used to facilitate safety assessments.

3.2.1. Introduction to graphs

Graphs are simple and intuitive mathematical entities used in a plethora of applications. In a graph, the *vertices* (or *nodes*) represent different elements or conditions, which are related to each other by *edges* (or *arcs*). If all vertices can eventually be reached from any other starting vertex, the graph is *connected*.

The nature of the connection varies between graphs. A classic use of graphs is representing rail connections. In Figure 3.1, vertices represent stations of South Holland, while edges indicate direct routes between them, which constitutes the relation represented in the graph. Even though the vertices of Figure 3.1 are approximately located as on a map, the shape and length of an edge and the positions of vertices are irrelevant for the sake of establishing if they are connected. In an *undirected* graph, if vertex v_i is connected to vertex v_j, then v_j is connected to v_i: the graph in Figure 3.1 is clearly undirected. In *directed* graphs, for which the above does not apply, edges can be either *inbound* or *outbound*; this is represented graphically by adding arrows to edges. A directed graph is *weakly connected*

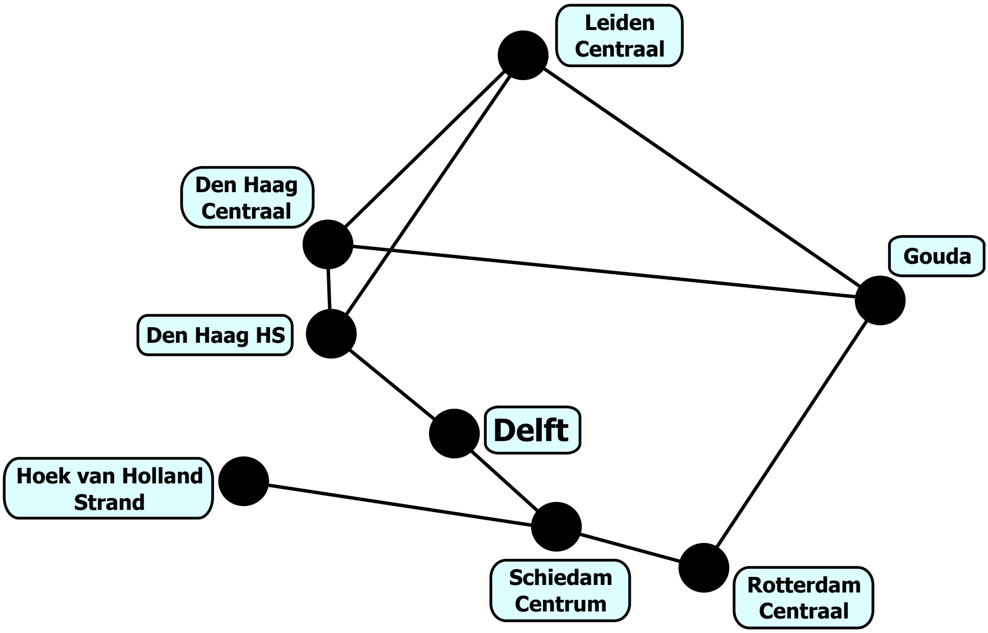


Figure 3.1: Undirected graph illustrating train connections between 8 stations in the railway system of South Holland.

if there is a path between any two vertices when the directed edges are replaced with undirected ones. See [97] for a more in-depth treatise of graph theory.

Among the structures that can numerically represent a graph, *adjacency matrices* are the most convenient for the applications of this chapter. These are square matrices with as many rows as the number of vertices. Assuming these are ordered, each entry A_{ij} of the matrix contains a one if the j^{th} vertex connects to the i^{th}, and a zero otherwise. Therefore, the j^{th} column indicates which vertices are connected to the j^{th} vertex; in general, the j^{th} column of the matrix obtained by raising the adjacency matrix to the power of $n \in \mathbb{N}$ indicates which vertices can be reached from the j^{th} vertex in exactly n transitions.

3.2.2. Operational envelope

A common definition of *flight envelope* describes it as those combinations of speed and load factor where the airplane can be safely controlled. Similarly to aircraft, other machineries and systems operate within a prescribed range of measurable conditions under which *performance* and *safety* can both be guaranteed. By analogy to the flight envelope, these constitute the *operational envelope* (OE) of the system.

In a manned system, an expert can monitor the system and can prevent violations of the OE so that operations are safe and efficient; however, if the system is unmanned, violations must be prevented automatically. Specifically, a RL policy that respects an OE is said to be *feasible*. It is important to notice that if the OE changes, a previously feasible policy might not be feasible anymore.

Consider now the safe exploration problem of Chapter 2, in which the RL agent must keep the system outside of an originally unknown FSS. Assuming that the FSS is stationary, the agent makes an underestimation of the SSS by means of *risk perception*, and selects actions so that the state is guaranteed to be safe. In this case, the OE is therefore the current underestimate of the SSS. Since every new estimate of the SSS will contain the previous as a subset, policies that are feasible at any point in time remain so with respect to new OEs. However, new feasible policies must still be found in flight in order to allow further exploration.

3.2.3. Graph generation: assumptions

The goal of this section is to illustrate how to rapidly generate a graph $\mathcal{G}$ whose vertices are states of the OE, and whose edges represent transitions between the states.

Let $\boldsymbol{x} \in \mathcal{S} \subseteq \mathbb{R}^n$ be the state of the system. The subset $\mathcal{S}_{env} \subseteq \mathcal{S}$ is the OE of the system and is assumed to be a finite, closed compact in the form

$$\mathcal{S}_{env} = [\underline{x}_1,\ \overline{x}_1] \times \cdots \times [\underline{x}_n,\ \overline{x}_n], \tag{3.1}$$

i.e., the envelope is given by all possible combinations of the individual components x_i of state $\boldsymbol{x} \in \mathcal{S}_{env}$ within a given interval $[\underline{x}_i,\ \overline{x}_i]$. It will be shown later how this assumption can be dropped.

As in chapter 2, given $\mathcal{D}$ the true dynamics of the system, it will be assumed that only a bounding model

$$\hat{\mathcal{D}} : \begin{cases} \dot{x}_1 \in \hat{\mathcal{D}}_1\ (x_1,\ x_2, \cdots,\ x_n,\ u) \\ \vdots \\ \dot{x}_n \in \hat{\mathcal{D}}_n\ (x_1,\ x_2, \cdots,\ x_n,\ u) \end{cases} \tag{3.2}$$

is available: note the membership relational operator $\in$ instead of the equivalence symbol. Finally, it will be assumed as usual that the action set $\mathcal{A}$ of the agent is fixed, and does not depend on current state and time.

Given the above assumptions, three discretization steps are necessary to obtain a directed graph whose vertices represent states of the system, and whose edges represent transitions between states. The first discretization is the selection of an appropriate *tiling* $\mathcal{T}$. This is obtained through an equispaced partition of the envelope $\mathcal{S}_{env}$ as

$$[\underline{x}_i,\ \overline{x}_i] = [\underline{x}_i,\ \underline{x}_i + \Delta_i] \ \cup\ [\underline{x}_i + \Delta_i,\ \underline{x}_i + 2\Delta_i] \ \cup \cdots \cup\ [\overline{x}_i - \Delta_i,\ \overline{x}_i] \tag{3.3}$$

where Δ_i is the tiling width for component x_i (see Figure 3.2). Each tile τ is then equal in size and equivalent to an interval of $\mathbb{R}^n$. The *index* of a tile is its "position" $\mathtt{i}(\tau) = (\mathtt{i}_1, \mathtt{i}_2, \ldots, \mathtt{i}_n)$ within the tiling, so that

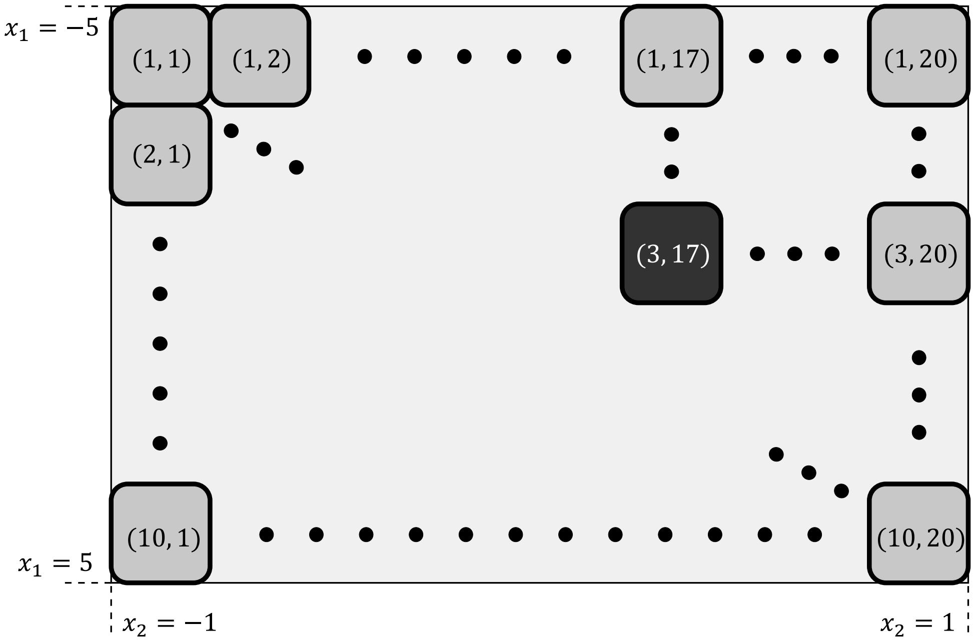


Figure 3.2: Tiling of the sample envelope $x_1 \in [-5, 5], x_2 \in [-1, 1]$. The tile with index $\mathtt{i} = (3, 17)$ is in black.

$$\boldsymbol{x} \in \tau \rightarrow x_i \in [\underline{x}_i + (\mathtt{i}_i - 1)\Delta_i,\ \underline{x}_i + \mathtt{i}_i\Delta_i]. \tag{3.4}$$

Each state belongs to one tile, with the exception of the zero-volume tiles contours. In various applications where partitions are applied, the size and shape of the tiles varies locally or even adaptively [98], or multiple tilings overlap as in tile coding [71]. The reason to apply only one equispaced $\mathcal{T}$ is that doing so greatly simplifies generating the graph, as it will become evident in the remainder of this section.

As a second discretization, the action set $\mathcal{A}$ is reduced to a "representative" subset $\mathcal{A}_{sub}$. To each pair (τ, u) of tiles and actions corresponds an edge in the graph. Thus, limiting the number of actions reduces the complexity of the graph and the number of computations needed to generate it.

The third and last discretization involves reducing the time-continuous dynamics of $\mathcal{D}$ to a discrete time equivalent:

$$\hat{\mathcal{D}}^{\Delta t} : \begin{cases} \Delta x_1 \in \hat{\mathcal{D}}_1^{\Delta t}\ (\tau \in \mathcal{T},\ u \in \mathcal{A}_{sub},\ \Delta t) \\ \vdots \\ \Delta x_n \in \hat{\mathcal{D}}_n^{\Delta t}\ (\tau \in \mathcal{T},\ u \in \mathcal{A}_{sub},\ \Delta t), \end{cases} \tag{3.5}$$

where Δt is the sampling time interval. The choice of which time interval to adopt can have a considerable effect on the accuracy of the graph when compared to the time-continuous dynamics. It is here assumed that the discrete dynamics of Eq. (3.5) are in interval form:

$$\hat{\mathcal{D}}_i^{\Delta t} = [\underline{\mathcal{D}_i^{\Delta t}}, \overline{\mathcal{D}_i^{\Delta t}}],\ i \in \{1, \dots, n\}. \tag{3.6}$$

As an example of the above discretizations, consider the double integrator with bounding model

$$\hat{\mathcal{D}} : \begin{cases} \dot{x}_1 \in \eta_1 x_2 \\ \dot{x}_2 \in -\eta_2 x_2 + \eta_3 u \end{cases} \tag{3.7}$$

and with given envelope $\mathcal{S}_{env} = [\underline{x}_1, \overline{x}_1] \times [\underline{x}_2, \overline{x}_2] = [-5, 5] \times [-1, 1]$ and action set $\mathcal{A} = [-1, 1]$. The uncertainty in the model is given by parameters $\eta_1 = [0.9, 1.1]$, $\eta_2 = [1, 1.2]$ and $\eta_3 = [0.8, 1]$. As a first step, an equispaced tiling with $\Delta_1 = 0.5$, $\Delta_2 = 0.1$ is chosen. The tile with index $\mathtt{i}(x) = (5, 17)$ is then the set

$$\{(x_1, x_2) \,|\, x_1 \in [-3, -2.5]\,;\ x_2 \in [0.6, 0.7]\}. \tag{3.8}$$

As a second step, a subset $\mathcal{A}_{sub} = \{-1, -\frac{1}{2}, 0, \frac{1}{2}, 1\}$ is adopted. Finally, a time interval $\Delta t = 0.125$ is chosen. The discretized bounding dynamics for this case would then be

$$\hat{\mathcal{D}}^{\Delta t} : \begin{cases} \Delta x_1 \in (\eta_1\, [\underline{x}_2 + \Delta_2(\mathtt{i}_2(\tau) - 1), \underline{x}_2 + \Delta_2 \mathtt{i}_2(\tau)])\, \Delta t \\ \Delta x_2 \in (-\eta_2\, [\underline{x}_2 + \Delta_2(\mathtt{i}_2(\tau) - 1), \underline{x}_2 + \Delta_2 \mathtt{i}_2(\tau)] + \eta_3 u) \Delta t. \end{cases} \tag{3.9}$$

3.2.4. Standard generation procedure

The graph $\mathcal{G}$ can now be generated as $|\mathcal{A}_{sub}|$ look-up matrices, indicating with $| * |$ the number of elements in set $*$. The look-up matrices act as the adjacency matrices of $\mathcal{G}$. Given action $u \in \mathcal{A}_{sub}$, initialize a square matrix with

$$\frac{\overline{x}_1 - \underline{x}_1}{\Delta_1} \cdot \frac{\overline{x}_2 - \underline{x}_2}{\Delta_2} \cdot \dots \cdot \frac{\overline{x}_n - \underline{x}_n}{\Delta_n}$$

rows, all of which are initially zeros. Consider then τ with index $(\mathtt{i}_1, \mathtt{i}_2, \dots, \mathtt{i}_n)$ and compute the intervals $\mathcal{D}_i^{\Delta t}(\tau, u)$. Note that since each τ is a multidimensional interval, one interval computation is sufficient for all states in τ, which motivates using rectangular tiles. However, this also means that additional uncertainty will be added to the transitions.

The minimum and maximum increments $\underline{\mathcal{D}_i^{\Delta t}}$ and $\overline{\mathcal{D}_i^{\Delta t}}$ determine which vertices are then reachable from τ. Indicating respectively the floor and ceiling operations with $\lfloor * \rfloor$ and $\lceil * \rceil$, τ will connect with all vertices with index range

$$\left(\left(\mathtt{i}_1 + \left\lfloor \frac{\underline{\mathcal{D}_1^{\Delta t}}}{\Delta_1} \right\rfloor\right) : \left(\mathtt{i}_1 + \left\lceil \frac{\overline{\mathcal{D}_1^{\Delta t}}}{\Delta_1} \right\rceil\right), \dots, \left(\mathtt{i}_n + \left\lfloor \frac{\underline{\mathcal{D}_n^{\Delta t}}}{\Delta_n} \right\rfloor\right) : \left(\mathtt{i}_n + \left\lceil \frac{\overline{\mathcal{D}_n^{\Delta t}}}{\Delta_n} \right\rceil\right)\right). \tag{3.10}$$

Algorithm 1 Graph generation

```
Initialize matrices, $\mathcal{A}_{sub}$, $\mathcal{T} \subset \mathbb{R}^n$
$A \leftarrow \mathcal{A}_{sub}$
while $|A| \neq 0$ do
    $u \leftarrow$ first element of $A$
    $T \leftarrow \mathcal{T}$
    while $|T| \neq 0$ do
        $\tau \leftarrow$ first element of $T$
        for $i = \{1, 2, \ldots, n\}$ do
            compute $\left\lfloor \frac{\underline{\mathcal{D}}_i^{\Delta t}}{\Delta_i} \right\rfloor$ and $\left\lceil \frac{\overline{\mathcal{D}}_i^{\Delta t}}{\Delta_i} \right\rceil$
            update current matrix
        $T \leftarrow T \setminus \{\tau\}$
    $A \leftarrow A \setminus \{u\}$
```

The corresponding entries in the matrix are then replaced with ones, indicating a possible state transition. This last passage is made possible by the adoption of an equispaced $\mathcal{T}$: for a "rectangular" but unevenly spaced tiling, each connection would need to be verified independently, increasing the amount of computations.

Algorithm 1 summarizes the procedure. Given $n_u = |\mathcal{A}_{sub}|$ and $n_\tau = |\mathcal{T}|$, the total complexity is then $\mathcal{O}(n \cdot n_u \cdot n_\tau \cdot \Delta)$, where Δ indicates the complexity of computing $\left\lfloor \frac{\underline{\mathcal{D}}_i^{\Delta t}}{\Delta_i} \right\rfloor$ and $\left\lceil \frac{\overline{\mathcal{D}}_i^{\Delta t}}{\Delta_i} \right\rceil$. While n and Δ depend on the environment, n_u and n_τ are a designer choice and can be altered to reflect the computational power of the UAV.

Consider again the double integrator example of Section 3.2.3. The graph $\mathcal{G}$ corresponding to the dynamics of Eq. (3.9) is obtainable as follows. Given the first action $u = -1$, a 4-dimensional matrix is generated with a total of $(20 \cdot 20)^2$ elements. These are initialized as zeros but later revised as soon as the transition are verified. For example, consider the aforementioned tile τ with index $\mathtt{i}(\tau) = (5, 17)$. It is then $\Delta x_1 \in ([0.9, 1.1] \cdot [0.6, 0.7]) \cdot 0.125 = [6.75, 9.625] \cdot 10^{-2}$ and $\Delta x_2 \in ([-1.2, -1] \cdot [0.6, 0.7] + [0.8, 1] \cdot -1) \cdot 0.125 = [-2.3, -1.75] \cdot 10^{-1}$. In terms of indexes, this is equivalent to an increase in index $\mathtt{i}_1$ between $\left\lfloor \frac{6.75 \cdot 10^{-2}}{0.5} \right\rfloor = 0$ and $\left\lceil \frac{9.625 \cdot 10^{-2}}{0.5} \right\rceil = 1$, and in index $\mathtt{i}_2$ between $\left\lfloor \frac{-2.3 \cdot 10^{-1}}{0.1} \right\rfloor = -3$ and $\left\lceil \frac{-1.75 \cdot 10^{-1}}{0.1} \right\rceil = -1$. Therefore, the zeros in the entries $(5, 17, 5, 14)$, $(5, 17, 5, 15)$, $(5, 17, 5, 16)$, $(5, 17, 6, 14)$, $(5, 17, 6, 15)$, $(5, 17, 6, 16)$ of the look-up matrix are replaced by ones.

It should be noted that one or more indexes to be replaced, according to Eq. (3.10), might be outside of the index range of the matrix. This means that the corresponding action leads to a violation of the OE, since it would reach a state outside of the envelope, and must therefore be prevented. In order to do so, no entries are replaced for that specific (τ, u) pair. In graphical terms, this is equivalent to not drawing the edge representing action u from τ.

It is also necessary to notice that the look-up matrices indicate connections but are not adjacency matrices themselves. Each entry of the adjacency matrices represents one edge, i.e., one exact transitions between vertices, while entries of the look-up matrices, taken individually, are not edges. In the previous example, action $u = -1$ in tile/vertex τ with index $\mathtt{i} = (5, 17)$, which in the adjacency matrix would take an entry, is represented in the look-up matrices by six entries, indicating a transition to multiple vertices. Therefore, the obtained $\mathcal{G}$ is a *hypergraph*. In graphical terms, this is equivalent to an edge separating into multiple arrows as represented in Figure 3.4. Which transition occurs when that action is selected from the agent in the actual environment depends on the true state and dynamics at that time.

Transitions between two tiles τ and τ' are therefore overapproximations of the actual transitions between states. This has a precautionary goal: if taking action u in $\boldsymbol{x}$ might cause a violation, the graph will prevent such action. However, this also means that the dynamics, as represented by $\mathcal{G}$, are artificially accelerated. Consider once again the above example. Computing the interval dynamics as given by Eq. (3.9) yields $\Delta x \in ([6.75, 9.625] \cdot 10^{-2}, [-2.3, -1.75] \cdot 10^{-1})$. The look-up matrix for the same action indicates that transitions $(5, 17, 5, 14)$, $(5, 17, 5, 15)$, $(5, 17, 5, 16)$, $(5, 17, 6, 14)$, $(5, 17, 6, 15)$, $(5, 17, 6, 16)$ are possible, which in terms of components is equivalent to $\Delta x \in ([0, 1] \cdot \Delta_1, [-3, -1] \cdot \Delta_2) = ([0, 0.5], [-0.3, -0.1])$. While Δx_2 is comparable between the two methods, Δx_1 as obtained with the graph formulation is increased several times with respect to the interval one. This effect is more pronounced for smaller Δt and for more coarse $\mathcal{T}$. Additionally, the difference tends to accumulate with the increase of the time horizon in the state space trajectory prediction. Figure 3.3 shows the predicted interval according to the graph dynamics (dark grey) and according to the interval dynamics (light grey) for three consecutive time-steps. It can be seen that, while the graph successfully overapproximates the dynamics, it also artificially accelerates the dynamics in x_1. A trade-off in terms of computational complexity, sampling time, and fidelity to the dynamics is therefore needed when designing the graph.

A final consideration concerns the assumption of Eq. (3.1) on the shape of the OE: it is possible to drop this assumption by adopting the following. First, find the least-volume set $\mathcal{S}^o$ in the form of Eq. (3.1) containing the OE. Second, find the index of those vertices $\tau^o : \tau^o \setminus \mathcal{S}_{env} \neq \emptyset$. Since these are at least partially out of the OE, actions that cause a transition to these are a possible violation. Therefore, matrix entries are not updated for these actions, just as illustrated above.

3.2.5. Accelerated generation procedure

Under certain assumptions, the previous procedure can be further accelerated. In Eq. (3.5), terms $\hat{\mathcal{D}}_i^{\Delta t}$ are functions of all components x_1, x_2, $\cdots$, x_n of $\boldsymbol{x}$. Let x_j' and x_j'', $x_j' \neq x_j''$, be two arbitrary values of the j^{th} component of $\boldsymbol{x}$. Then the i^{th} component x_i of $\boldsymbol{x}$ is said to be *independent* of x_j if

$$\hat{\mathcal{D}}_i(x_1, \dots, x_j', \dots, u) = \hat{\mathcal{D}}_i(x_1, \dots, x_j'', \dots, u). \tag{3.11}$$

Considering again the example given by the dynamics of Eq. (3.7), it can be seen

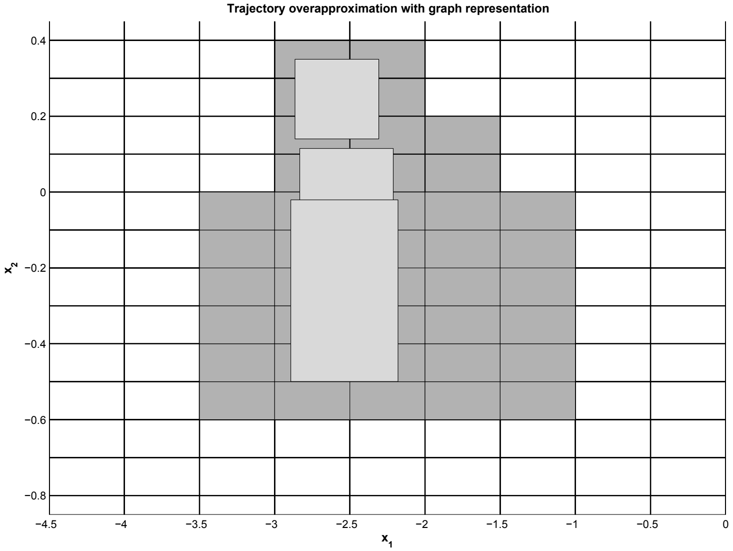


Figure 3.3: A comparison between the interval and the graph representation of the dynamics after three time-steps. The graph-yielded trajectory (dark grey) bounds the one generated by the original interval dynamics, at the cost of overestimating the dynamics of component x_1.

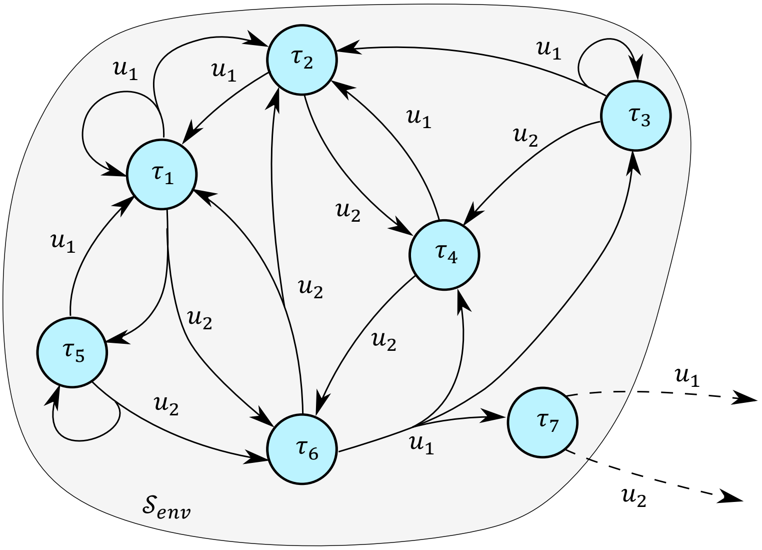


Figure 3.4: Example of a hypergraph. From each of the seven vertices τ_i depart two edges ($\mathcal{A}_{sub} = \{u_1, u_2\}$) with the exception of τ_7, since they would result in a violation of $\mathcal{S}_{env}$.

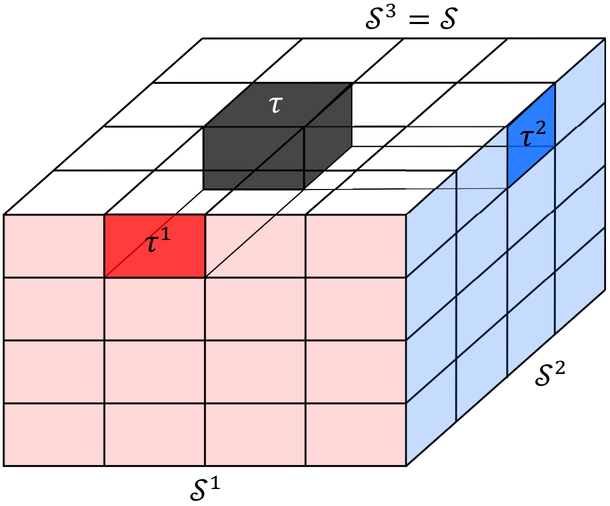


Figure 3.5: Envelope $\mathcal{S}$ presents three projected envelopes $\mathcal{S}^1$ (in pink), $\mathcal{S}^2$ (in light blue) and $\mathcal{S}^3 = \mathcal{S}$. Tile τ (in black) corresponds to projected tiles τ^1 (in red) and τ^2 (in blue).

then that component x_2 of $\boldsymbol{x}$ is independent of component x_1. Indicating by $\mathtt{ind}_i$ the set of components from which x_i is independent, define now the *projected envelope* $\mathcal{S}^i$ as

$$\mathcal{S}^i = \prod^{\times}_{j \notin \mathtt{ind}_i} [\underline{x}_j,\ \overline{x}_j] \tag{3.12}$$

where the apex "×" indicates the set product. It is then

$$\hat{\mathcal{D}}^{\Delta t} : \begin{cases} \Delta x_1 \in \hat{\mathcal{D}}_1^{\Delta t}\,(\mathcal{S}^1,\ u,\ \Delta t) \\ \qquad\vdots \\ \Delta x_n \in \hat{\mathcal{D}}_n^{\Delta t}\,(\mathcal{S}^n,\ u,\ \Delta t). \end{cases} \tag{3.13}$$

E.g., suppose that $\hat{\mathcal{D}}_i^{\Delta t} = \hat{\mathcal{D}}_i^{\Delta t}(x_2, x_3, x_5, u, \Delta t)$. Then $\mathcal{S}^i = [\underline{x}_2,\ \overline{x}_2] \times [\underline{x}_3,\ \overline{x}_3] \times [\underline{x}_5,\ \overline{x}_5]$. An example of components that are independent of each other can be obtained when separating the longitudinal and lateral dynamics of an aircraft.

The tile partition of Eq. (3.3) can be applied to projected envelopes as well, obtaining equispaced projected tilings $\mathcal{T}^i$ composed of "projected tiles" τ^i (see Figure 3.5). As a result, Eq. (3.13) permits a faster graph computation with respect to the one previously illustrated. Algorithm 2 illustrates the procedure. Given action $u \in \mathcal{A}_{sub}$, for each projected tile τ^i of $\mathcal{T}^i$ compute the minimum and maximum advancement in index $\left\lfloor \frac{\underline{\mathcal{D}}_i^{\Delta t}}{\Delta_i} \right\rfloor$ and $\left\lceil \frac{\overline{\mathcal{D}}_i^{\Delta t}}{\Delta_i} \right\rceil$. This is equivalent to defining a "semigraph" function

$$g^i : \mathcal{S}^i \times \mathcal{A}_{sub} \to \mathbb{N} \tag{3.14}$$

Algorithm 2 Graph generation with semi-graphs

```
**Initialize** matrices, $\mathcal{A}_{sub}$, $\mathcal{T}^i$, $i = 1, \dots, n$
$A \leftarrow \mathcal{A}_{sub}$
**while** $|A| \neq 0$ **do**
    $u \leftarrow$ first element of $A$
    **for** $i = \{1, 2, \dots, n\}$ **do**
        $T^i \leftarrow \mathcal{T}^i$
        **while** $|T^i| \neq 0$ **do**
            $\tau^i \leftarrow$ first element of $T^i$
            compute $\left\lfloor \frac{\underline{\mathcal{D}}_i^{\Delta t}}{\Delta_i} \right\rfloor$ and $\left\lceil \frac{\overline{\mathcal{D}_i^{\Delta t}}}{\Delta_i} \right\rceil$
            **for** $j = \{1, 2, \dots, \frac{|\mathcal{T}|}{|\mathcal{T}^i|}\}$ **do**
                update current matrix
            $T^i \leftarrow T^i \setminus \{\tau^i\}$
    $A \leftarrow A \setminus \{u\}$
```

which associates to each pair (τ^i, u) the corresponding index increment. The obtained increments are then applied to all tiles that "project" on τ^i.

Depending on Δ, the complexity of the procedure is either $\mathcal{O}(n_u \cdot n \cdot n_\tau)$ or $\mathcal{O}(n_u \cdot n \cdot n_{\tau^i} \cdot \Delta)$, where n_{τ^i} indicates the average number of tiles in $\mathcal{T}^i$. It can be seen that the complexity of this procedure is strictly not worse than the one indicated by Algorithm 1, and can be significantly better depending on Δ and n_{τ^i}.

3.2.6. Graph representations in reinforcement learning

This conclusive section illustrates how the obtained graph formulation of the dynamics can be employed to increase the online efficiency of both model based and safe RL methods.

A similarity can be drawn between a graph $\mathcal{G}$ and a Markov decision process (MDP), which constitutes the usual framework for RL[1]. Both mathematical structures indicate time-discrete transitions between a finite set of environment conditions, caused by the implementation of one among a finite number of actions. In addition to similarities, differences can be found as well. The vertices of $\mathcal{G}$ do not directly correspond to the states of the environment, but represent sets of states. Furthermore, $\mathcal{G}$ is an hypergraph which, due to accumulated uncertainties, overapproximates the dynamics. Conversely, an MDP indicates transitions between crisp states, and these transitions, which can be stochastic, are not overapproximated. As a result, a graph $\mathcal{G}$, being similar but not identical to an MDP, cannot be used in place of the latter for model based RL. However, the graph representation could be utilized to obtain a stochastic approximation of the underlying MDP of the environment.

[1] In the following, canonical Markov decision processes, which admit a finite number of discrete states and actions, are considered. This framework can be adapted to accommodate continuous systems by an appropriate state and action discretization.

The most immediate contribution of the graph representation, however, comes from its application to the *challenge of safety*, and specifically to the online computation of backups. In accordance to Section 2.3.3, two conditions must be satisfied for a control sequence $u(k), u(k+1), \dots, u(k+m-1)$ to be a backup:

1. $[\boldsymbol{x}_{k+j}] \in \mathrm{SSS}$, $j = 0, 1, \dots, m$;
2. $\forall \boldsymbol{x}_{k+m} \in [\boldsymbol{x}_{k+m}]$, $\exists \boldsymbol{x}_p$ s.t. $(x_{k+m} - \boldsymbol{x}_p) \in [\epsilon]$, or $\exists \boldsymbol{x}_E$ s.t. $(x_{k+m} - \boldsymbol{x}_E) \in [\delta]$.

Therefore, the agent must compute the intervals $[\boldsymbol{x}_{k+j}]$, check if they are contained within the known SSS, and evaluate whether or not they are in a given neighborhood of at least one visited state $\boldsymbol{x}_p$ or equilibrium point $\boldsymbol{x}_E$. Since multiple checks might be required, this assessment can be computationally demanding.

Using a graph representation of the dynamics reduces this burden. Before exploration, the look-up matrices can be converted into look-up tables $\mathcal{T} \times \mathcal{A}_{sub} \to \mathcal{T}$ indicating for each vertex and action the corresponding edge transitions. Then, if the environment is in state $\boldsymbol{x}(k)$ an overapproximation of the interval containing $\boldsymbol{x}(k+1)$ for action $u(k)$ can be obtained by consulting the corresponding look-up table for $\tau(k) \ni \boldsymbol{x}(k)$. The yielded vertices overapproximate the desired interval. Furthermore, the look-up tables can be applied iteratively to further predict the backup trajectory, given a backup sequence $(u(k+1), \dots, u(k+m-1))$. Additionally, the backup conditions can be conveniently expressed in terms of vertices rather than states. At the start of exploration, given an initial SSS, it is relatively straightforward to identify those vertices for which $\tau \cap \mathrm{FSS} = \emptyset$. Similarly, it is possible to identify those vertices that are included within a neighborhood of an initial set of visited and equilibrium states. These lists of vertices can be progressively updated during exploration, as the SSS expands and new states are visited. Alternatively, the OE can be made to coincide with the known SSS, and a new graph can be computed periodically as the SSS increases in size.

Therefore, backups can be validated more rapidly by adopting a graph representation. However, the fact that the collection of vertices obtained through $\mathcal{G}$ overapproximates the intervals $[\boldsymbol{x}_{k+j}]$ has the unwanted effect of making the backup conditions more stringent.

3.3. Graph pruning

The previous section introduced the concept of OE, presented a procedure to generate graph representations of the dynamics, and explained how these structures can be used to ease the computational demands of safe RL and in particular of backup searches. This section presents *graph pruning*, another method that can provide safety of exploration for RL agents. This method achieves safety not through the definition of a backup, but by a-priori constraining the set of eligible policies of the RL agent.

3.3.1. Pruning the graph

As mentioned in Section 3.2.4, if an action u in vertex τ determines a violation of the envelope, the corresponding edge is not added to $\mathcal{G}$, and the look-up matrix cor-

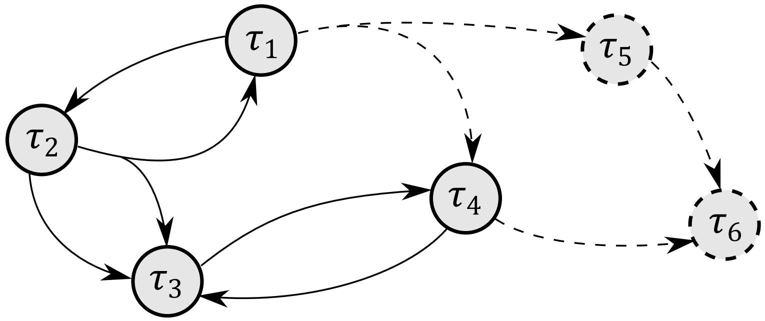


Figure 3.6: An example of graph pruning. Initially one sink (τ_6) is found. As a consequence of the removal of the inbound edges, another vertex (τ_5) becomes a sink. The subsequent removal of the inbound split edge does not lead to the generation of new sinks.

responding to u is not updated. By doing so, violating actions are directly excluded from the graph. However, this is not sufficient to guarantee safety. For example, in Figure 3.4, vertex τ_7 has no outbound edges. In graph theory, these vertices τ_s are indicated as *sinks*. These originate during graph generation if, $\forall u \in \mathcal{A}_{sub}$, $\exists \boldsymbol{x} \in \tau_s$ such that u causes a violation of the OE, as is the case for action u_1 and u_2 in τ_7 of Figure 3.4. If the environment is in a sink, all the agent's actions are then potentially unfeasible. It is important to recall that, in the problem formulation, the agent cannot refrain from taking actions, as even passive actions, e.g., "do nothing", are part of the set $\mathcal{A}$. For this reason, the agent should prevent visiting sinks.

Consider now vertex τ with an outbound edge (τ, u_1) which is inbound to a sink, and an edge (τ, u_2) that is not. Due to the overapproximating nature of $\mathcal{G}$, taking action u_1 does not mean that the environment will surely transition into a sink vertex; however, if u_2 is performed, the environment is guaranteed not to transition into a sink. Therefore, in order to prevent reaching a sink, it is sufficient to *prune*, i.e., to remove from $\mathcal{G}$, all edges that are inbound to a sink, as well as the sink themselves. Referring to Definition 2.3, this is equivalent to guarantee that the system will not enter an LTF state. Notice, however, that pruning might result in new sinks being created. In turn this require additional pruning. These two steps - sink detection and pruning - alternate until no new sinks are identified. The process is illustrated in Figure 3.6.

The pruning and sink detection procedures are illustrated in Algorithm 3. Indicating by $\mathtt{T}_j$ the look-up matrix corresponding to the j^{th} action, vertex τ with index $\mathtt{i}(\tau) = (\mathtt{i}_1, \dots, \mathtt{i}_n)$ is a sink if $\forall j \in \{1, \dots, |\mathcal{A}_{sub}|\}, \forall \tau' \in \mathcal{T}$,

$$\mathtt{T}_j(\mathtt{i}_1, \dots, \mathtt{i}_n, \mathtt{i}'_1, \dots, \mathtt{i}'_n) = 0 \tag{3.15}$$

where $\mathtt{i}(\tau') = (\mathtt{i}'_1, \dots, \mathtt{i}'_n)$, so that $\mathtt{T}_j(\mathtt{i}_1, \dots, \mathtt{i}_n, \mathtt{i}'_1, \dots, \mathtt{i}'_n)$ is the entry of $\mathtt{T}_j$ indicating the connection between τ and τ'. If τ is found to be a sink, the look-up matrices must be appropriately pruned. This is done by replacing all entries $\mathtt{T}_j(\dots, \mathtt{i}_1, \dots, \mathtt{i}_n)$ which connect any vertex to τ with zeros, $\forall j \in \{1, \dots, |\mathcal{A}_{sub}|\}$. At the end of the process, when sink detection does not reveal any new sinks, the pruned graph $\mathcal{G}_p$

Algorithm 3 Graph pruning

```
Initialize look-up matrices $\mathtt{T}_j$, $\mathcal{T}$, $\mathcal{A}_{sub}$
sinks ← $\emptyset$
while graph is not pruned do
    $T \leftarrow \mathcal{T} \setminus$ sinks
    while $|T| \neq 0$ do
        $\tau \leftarrow$ first element of $T$
        for $j = \{1, 2, \dots, |\mathcal{A}_{sub}|\}$ do
            $T' \leftarrow \mathcal{T}$
            while $|T'| \neq 0$ do
                $\tau' \leftarrow$ first element of $T'$
                if edge between $\tau$ and $\tau'$ then
                    $\tau$ is not a new sink
                $T' \leftarrow T' \setminus \{\tau'\}$
        if $\tau$ is a sink then
            update matrices $\mathtt{T}_j$
        $T \leftarrow T \setminus \{\tau\}$
    update sinks
    if no new sinks then
        graph is pruned
```

can be

- connected or weakly connected;
- not connected;
- the *null graph*, i.e., a graph with zero vertices.

In either of the first two cases, $\mathcal{G}_p$ can be interpreted as a representation of all policies that are guaranteed to be safe, i.e., feasible with respect to the OE. Therefore, $\mathcal{G}$ can be used to ensure that the current policy of the agent is safe, regardless of how the policy is obtained or updated (e.g., policy search). A (stochastic) policy $\pi : \mathcal{S}_{env} \times \mathcal{A}_{sub} \to [0, 1]$, is feasible only if $\forall \boldsymbol{x} \in \mathcal{S}_{env}$, $\boldsymbol{x} \in \tau$, $\pi(\boldsymbol{x}, u) \neq 0$ means that the edge (τ, u) is in $\mathcal{G}_p$. In the third case, $\mathcal{G}$ cannot be used to guarantee that the current policy is safe; however, this does not necessarily mean that a safe policy does not exist. A refinement of either $\mathcal{T}$ or $\mathcal{A}_{sub}$, which reduces the overapproximation of the graph, might eventually lead to a solution.

3.3.2. Related work

To prevent the state of a controlled system from abandoning a region of the state space is not a new problem in control theory; therefore, other approaches and techniques exist in literature for achieving this objective. A purely mathematical approach is the computation of *invariant sets* or *viable sets*, i.e., a set of states where the system can stay indefinitely. Feuer and Heymann [99] investigate properties of such sets for linear systems. Oishi et al. [100] apply *Reachability Analysis* to determine the flight envelope of a jet aircraft, and for planar collision avoidance between two aircraft. Both works offer an integral exhaustive approach to the problem. Alternatively, energy based approaches have been investigated. Khatib [89] provides an application of *Artificial Force Fields* to manipulators and mobile robots. With this method, applied control forces generate an appropriate potential field, so that the system is forbidden from reaching undesirable states. Fraichard and Asama [87] also consider the problem of collision avoidance for autonomous agents, relying on a sensor-based approach to achieve *Safe Motion Planning*. Gillula and Tomlin [101] show an approach for UAVs with a combination of Reachability Analysis and *reinforcement learning* that prevents the violation of safety constraints under noise and a successful application [102] to a quadrotor. Kwatny and Allen [103] apply the problem to the computation of the safe flight envelope for an impaired aircraft. This is done by individuating its original *trim points*, and investigating how to switch to a different trim point when the current one vanishes due to the occurrence of impairment.

These approaches are either very specific, relying on particular features of the system at hand, or computationally heavy. Graph pruning differs from the above methodologies in that it is both computationally advantageous, since it employs the graphical representation of the dynamics given by $\mathcal{G}$, and applicable to different vehicles and tasks, as long as the task and the desired conditions can be represented as an OE.

3.3.3. Application to a UAV navigation task

In this section, the graph pruning method is applied in simulation to a UAV flying inside a corridor. The goal of the agent is to advance towards the end of the corridor while preventing a collision with the walls. In order to accomplish this goal:

1. an OE compatible with the task is defined;
2. given a bounding model of the environment, a graph $\mathcal{G}$ is generated;
3. the pruned graph $\mathcal{G}_p$ is obtained.

Afterwards, any arbitrary policy could be applied as long as it is compatible with the graph; in this example, the policy consists in randomly selecting an action among those allowed by $\mathcal{G}_p$.

As already stated, the first step is to define an OE as to avoid collisions while navigating towards the exit of the corridor. Thus, the envelope will be defined with respect to $\hat{d}_1$ and $\hat{d}_2$, estimated distances respectively from the left and the right wall, and to the estimated direction of the corridor $\hat{\psi}_c$ (see Figure 3.7). A collision occurs if either $d_1 < 0$ or $d_2 < 0$. The agent estimates the rate of change of the two distances as

$$\hat{\dot{d}}_1 = -\hat{\dot{d}}_2 \in \hat{V} \cdot \sin(\psi - \hat{\psi}_c), \tag{3.16}$$

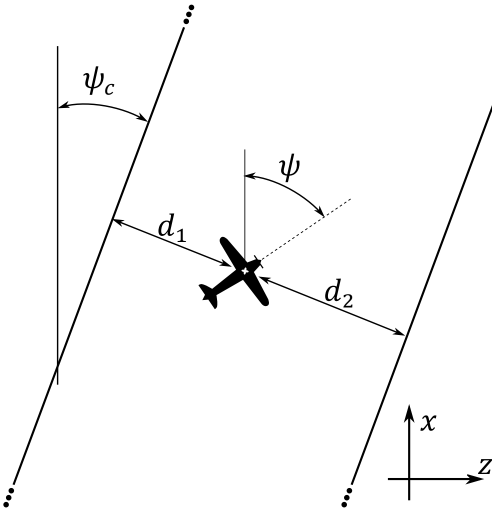


Figure 3.7: The corridor task, with distances d_1 and d_2 of the UAV from the walls, heading ψ and corridor direction ψ_c. Coordinates x and z indicate absolute position.

where the first equivalence is due to assuming a constant width of the corridor, and where $\hat{V}$ is an estimate of the constant speed of the UAV and ψ is the current heading. The uncertain UAV dynamics given by

$$\begin{aligned} \dot{z} &\in \hat{V} \cdot \sin(\psi), \\ \dot{x} &\in \hat{V} \cdot \cos(\psi), \\ \dot{\psi} &\in \hat{\dot{\psi}}_{max} \cdot u \end{aligned} \tag{3.17}$$

are assumed, where x and z identify the position of the UAV, and $\hat{\dot{\psi}}_{max}$ is an interval estimating the maximum turning rate of the UAV, which constitutes the agent control action. The OE $\mathcal{S}_{env}$ is then selected as

$$\mathcal{S}_{env} = \left\{d_1, \psi \mid d_1 \in [0, w], \psi \in \left[\hat{\psi}_c - 20°, \hat{\psi}_c + 20°\right]\right\}, \tag{3.18}$$

with w the width of the corridor, so that the bounding model of Eq. (3.2) for the state $\boldsymbol{x} = (d_1, \psi)$ is given by Eq. (3.16) and Eq. (3.17). The arbitrary constraint $\psi \in [\hat{\psi}_c - 20°, \hat{\psi}_c + 20°]$ is introduced to guarantee that the UAV actually flies from one end of the corridor to the other.

The turning rate and speed of the UAV are initialized to the agent as $\hat{\dot{\psi}} = [5°, 10°]$ and $\hat{V} = [1, 1.5]\frac{\mathrm{m}}{\mathrm{s}}$. Width w and estimated direction $\hat{\psi}_c$ are provided to the agent, defining the OE.

As a second step, the graph $\mathcal{G}$ is generated: a tiling $\mathcal{T}$ with $\Delta_1 = \frac{w}{20}$ and $\Delta_2 = \frac{40°}{20} = 2°$ is selected; a subset $\mathcal{A}_{sub} = \{-1, 0, 1\}$ is chosen; finally, $\Delta t = 0.5s$ is adopted. The hypergraph $\mathcal{G}$ is then generated and pruned. If the pruning results

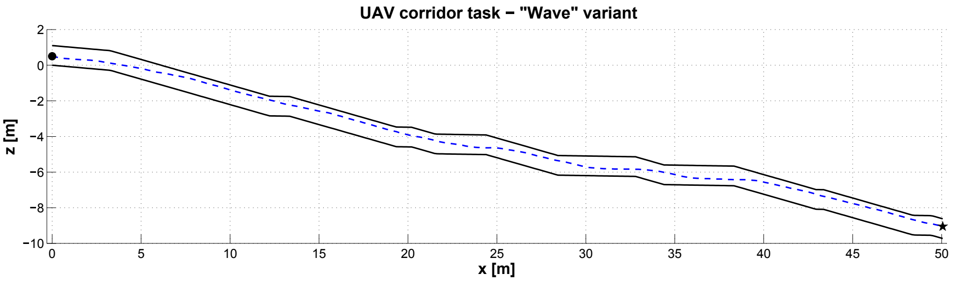


Figure 3.8: A typical typical trajectory for the "wave" variant. The dot and the star represent the UAV initial and final position. The UAV trajectory (dashed line) complies with the corridor walls (solid lines) so that a collision is avoided.

in a null graph, the following strategy is proposed: rather than refining $\mathcal{T}$ or $\mathcal{A}_{sub}$, the speed is halved as

$$\hat{V} \leftarrow 0.5 \cdot \hat{V}. \tag{3.19}$$

This choice is motivated by assuming that reducing the speed also decreases the difficulty of the task.

At the start of each simulation, V and $\dot{\psi}_{max}$ are assigned a randomly selected value from the intervals $\hat{V}$ and $\hat{\dot{\psi}}$. Simulations are initialized in a random state $\boldsymbol{x}_0$, with the condition that vertex $\tau_0 \ni \boldsymbol{x}_0$ is not a sink of $\mathcal{G}_p$. A stochastic policy is then enforced by randomly selecting the action corresponding to one of the available edges for the current vertex. A new state is then generated according to

$$\begin{aligned} \dot{z} &= V \cdot \sin(\psi), \\ \dot{\psi} &= \dot{\psi}_{max} \cdot u, \\ \dot{d}_1 &= V \cdot \sin(\psi - \psi_c). \end{aligned} \tag{3.20}$$

Given the above initialization, this section presents three variants of the task.

Wave variant

In the first "wave" variant, the UAV follows a corridor of constant width w. The direction of the corridor at the entrance, $\psi_c = \psi_{c0}$, is assigned randomly and is initially constant. The UAV is assigned an estimate $\hat{\psi}_c$ equal to this value. Given the OE indicated by w and $\hat{\psi}_c$, $\mathcal{G}$ is generated to be later used to obtain the aforementioned stochastic policy. However, at each time-step, there is a 10% probability that the direction of the corridor changes according to a random rate $\dot{\psi}_c \in [-\pi/2, \pi/2]\ \mathrm{s}^{-1}$, with the constraint $||\psi_{c0} - \psi_c|| \leq 20°$. The estimate $\hat{\psi}_c = \psi_{c0}$ is therefore inaccurate during the flight; however, the UAV agent still identifies correctly the distance from the wall d_1.

Figure 3.8 shows a typical result for this variant. To generate the graph, up to four subsequent graph generations are necessary, with each generation requiring

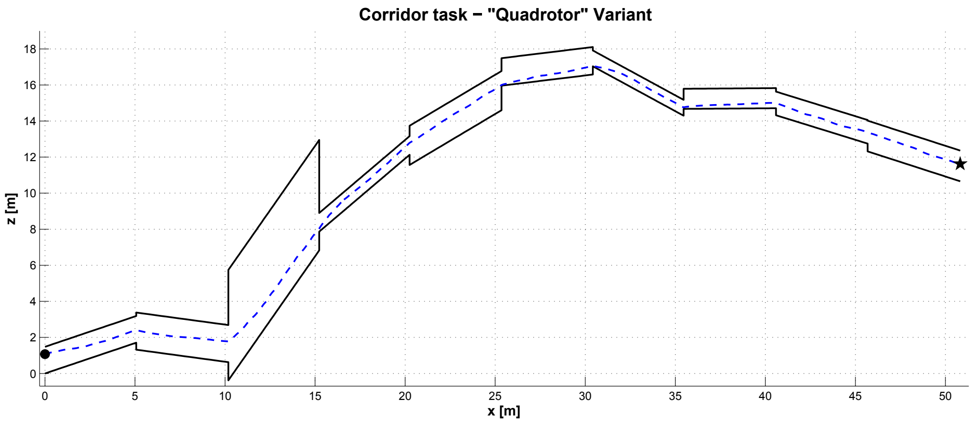


Figure 3.9: A typical trajectory for the "quadrotor" variant. The dot represent the UAV initial position. The UAV trajectory (dashed line) complies with the corridor walls (solid lines) so that a collision is avoided, and reaches the final position represented by the star.

around a second[2]. The stochastic policy obtained from the pruned graph is collision-free. Even as the corridor is the steepest, i.e., $\|\psi_{c0} - \psi_c\| = 20^\circ$, the UAV flies very near the walls but does not collide.

Quadrotor variant

In the second "quadrotor" variant, the corridor is divided in segments of varying length, with direction and width determined randomly. Given the width w_0 and direction ψ_{c0} of the first segment of corridor, the UAV agent generates and prunes an initial graph, the policy of which is collision-free. However, the previous OE is not valid in new segments, which have a different value of w and ψ_c. The graph $\mathcal{G}_p$ and the corresponding policy are therefore not safe. In this task, it is assumed that the UAV agent can notice this discrepancy, and then compute a new OE and a new graph on the spot, if necessary adjusting its heading and velocity.

Figure 3.9 shows a typical execution of this variant. Adopting the appropriate OE and reducing speed, the agent manages to avoid collision even for narrow sections, e.g., the fourth, where the corridor is only $0.65\mathrm{m}$ wide. Between two and four graph generations, depending on the width of the corridor, are needed to obtain this policy. However, in order to do so significant changes in flight direction are necessary, and the obtained trajectory is not smooth. Therefore, it assumed that the agent controls a UAV which can hover in place, e.g., a quadrotor or a flapping wing MAV.

Shrinking variant

The third "shrinking" variant involves those UAVs that are not able of hovering in place, and for which there are time constraints on the computation of a new OE. In this variant, the corridor is once again divided in segments of different width and direction. For the first segment it is $w_0 = 3\mathrm{m}$; for each segment thereafter, width

[2] on an Intel Core i5-3360M CPU, 2.80GH

reduces as $w|_{k+1} = w|_k - 0.5\mathrm{m}$; additionally, the direction of the segment changes as $\psi_c|_{k+1} = \psi_c|_k - 5°$. The agent uses these estimates to obtain an OE and a graph, but as in the previous variant, a new envelope is needed for each segment. Assuming that the UAV cannot hover in place and freely rotate, in this variant the agent adopts a different procedure to reuse part of its previous knowledge.

Due to the shrinking of the corridor, some vertices of $\mathcal{G}_p$ that were previously feasible will now lead to collision. However, if these are pruned from the graph, a new collision-free policy can be found. Therefore, at the start of a new segment the graph is pruned again as described in Section 3.3.1, starting from these vertices. In the meantime, the agent temporarily relies on the previous policy to generate actions. The reason to adopt this solution is that, if the new environment (and thus the new OE) is similar to the previous, pruning a second time is faster than generating a new graph from the start. Additionally, while it cannot be guaranteed that the in-pruning flight is collision-free, this is more likely the smaller the difference between the previous and the new environment. As soon as $\mathcal{G}_p$ is further pruned, and assuming the result is not a null graph, this can be used to infer a policy which is collision-free, but possibly inefficient. For this reason, given the new values of ψ_c and w, an entirely new graph $\mathcal{G}_{new}$ is still generated and pruned, from which a final policy is then extracted.

Figure 3.10 represents the task in its "shrinking" variant. The segments of corridor change in width from an initial value of $3\mathrm{m}$ to a final value of $0.5\mathrm{m}$. Results show that in the last segments of the corridor it is difficult to obtain a feasible policy: the agent manages to do so in 61% of the runs. The percentage of success increases to 85% if only the first four segments are considered. After generating $\mathcal{G}_{new}$, the agent locates the vertex containing its current state. If this has been pruned, then there is no action available to the agent, and the policy cannot be defined for the current state of the UAV. The agent therefore fails to guarantee that the envelope will not be violated, although this not necessarily implies that a collision is inevitable.

An explanation for the increased rate of failure can be found in the constant reduction in width. Between the first and second segment of the corridor, the relative reduction of width $\frac{w|_k - w|_{k+1}}{w|_k}$ is 10%, quite moderate when compared to the relative reduction between the third and fourth segments, equal to 33%, and between the fourth and fifth, which is 50%. This stands as a confirmation that the method is more efficient when the changes in environment are gradual.

3.4. Conclusions

This chapter investigates the use of graph representations to address the computational complexity of backup generations, and more generally of reinforcement learning agents in unsafe environments, for which the challenge of *online efficiency* arises. Being simple mathematical entities that can be stored and manipulated with ease, graphs accommodate the need for efficiency, and can be applied to different environments. Additionally, the chapter presents procedures to compute and modify these graphs online, due to their limited complexity.

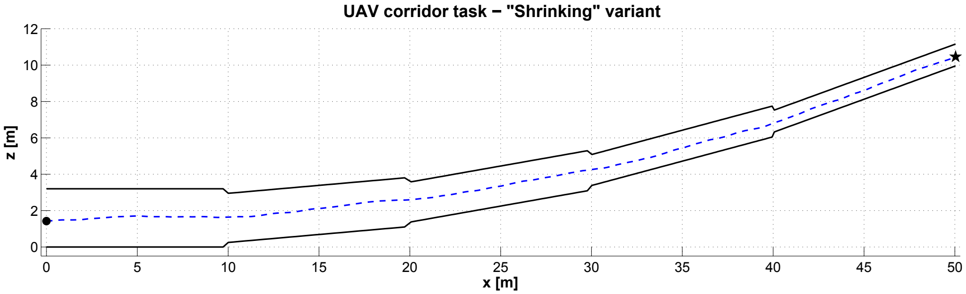


Figure 3.10: The UAV trajectory of a successful run of the "shrinking" variant of the task. The UAV starts at the left end end of the corridor in the position indicated by the dot. The corridor shrinks of $0.5\mathrm{m}$ and bends of 5° at every segment. The UAV moves from one segment to the next by relying on an further pruned version of its previous graph, and afterwards computing a new graph for the new envelope.

Furthermore, the chapter introduces an approach that verifies if the RL agent policy is provably feasible. This method relies on the concept of pruning, which consists in eliminating those vertices that might lead to a violation. This is equivalent to preventing the agent from visiting the Lead-to-Fatal states (Eq. (2.3)) of the environment. The method is tested on a simulated UAV application. Simulation results show that the method is effective in preventing collisions when the envelope is stationary. When the envelope is dynamic, the approach can be applied on the condition that the changes are gradual, so that the previous pruned graph is not entirely invalidated.

The methods introduced in this chapter serve as a prerequisite for the methods of Part II. Specifically, Chapter 5 illustrates how the look-up tables can be used to compute and verify backups, similarly to the OptiSHERPA algorithm, but reducing the effort of computing the interval dynamics. The Vertex Classification method of Chapter 6 consists in assigning weights indicating safety to each vertex, which can be used to evaluate the feasibility of a policy, similarly to graph pruning.

4

Hierarchical methods

This chapter investigates the advantages of hierarchically structured agents for the sake of safe exploration. The chapter shows that hierarchy can mitigate the ***challenge of robustness*** *in unsafe and uncertain environments, by abstracting the state, by embedding design knowledge, and by restricting the set of discoverable policies. Furthermore, the chapter presents a specific method denominated* ***Virtual Safety Training*** *which allows the agent to learn safe policies in a projected environment. This method is shown to be successful in providing a safe initial policy for the agent and in reducing fatal occurrences.*

4.1. Introduction

Chapter 3 presented an effective and generalized methodology to address the *challenge of online efficiency*, reducing the computational effort and simplifying backup search and policy safety assessment. The method consists of reducing the operational envelope (OE) of the agent into a tiling, propagate a bounding model, and using tiles as vertices of a graph.

Coarser tilings, whose refinement depends on the memory of the UAV, produce more imprecise overapproximations of the bounding dynamics than more refined ones. This can be problematic when applying graphs to state spaces that are high-dimensional or extensive in size, and that are uncertain, in terms of dynamics and safe states. In these cases, pruning is likely to result in a null graph, and backups with more than a few time-steps tend to become imprecise and unreliable. Additionally, environments with uncertainties suffer from the *challenge of robustness*. The more the environment dynamics are uncertain, the less model-based policies are reliable. Also, the more the SSS is uncertain, the harder it is to define OEs. Therefore, depending on the amount of uncertainty, the reduced computational complexity of graph methods might come at the cost of safety.

This chapter confronts the above problems, and in particular the *challenge of robustness*. Hierarchical Reinforcement Learning (HRL) is implemented to increase safety of exploration, to mitigate the effects of environment uncertainty, and to reduce the size and the dimensionality of the state space. This application of hierarchy to safe exploration is introduced under the new field of Safe Hierarchical Reinforcement Learning (SHRL). Additionally, an off-line learning procedure denominated Virtual Safety Training (VST) is proposed, which constitutes the main contribution of the chapter. By adopting HRL architectures, the agent can accommodate more uncertain, higher dimensional environments, thus increasing robustness of the learning process.

The chapter is structured as follows. Section 4.2 introduces HRL and examines the properties of hierarchy within the field of SHRL. Section 4.3 presents VST as a specific SHRL method, together with the necessary assumptions for its application. Section 4.4 shows an application of VST to a classical, discrete state problem, where an agent is tasked with finding a goal position in a maze environment. A more complex MAV navigation task is discussed in Section 4.5. In this section, uncertain, continuous dynamics and non-exhaustive beliefs are considered. Finally, Section 4.6 concludes the chapter.

4.2. Hierarchical Reinforcement Learning

This section introduces the central concept of *temporally extended actions* (TEAs) for HRL, as well as a few notable HRL algorithms. Afterwards, the advantages of HRL agents to safe exploration are presented.

4.2.1. Introduction to HRL

A well known issue with RL is that, at the start of learning, the policy is usually inefficient since the agent's knowledge of the environment is incomplete. Extensive exploration can be necessary before a task is correctly performed, depending on its complexity: the *curse of dimensionality* dictates that learning times grow exponentially with the number of states, which poses a problem when considering large and discrete, or continuous state spaces. This initial "blind search" can limit the applicability of RL algorithms, for which keeping learning time to a minimum is therefore a strong requisite.

HRL is among the methodologies adopted to reduce the curse of dimensionality and the "blind search". In classic or "flat" RL, the agent's actions have a one-step duration: after action u, the agent observes the new state, possibly updates its value function, and immediately decides on a follow-up action. These one-step actions are said to be *primitive*. HRL agents are different from flat RL agents in that they are allowed to execute more than one primitive action between each decision step. This sequence of primitives is a temporally extended action (TEA). Which and how many primitives each TEA contains depends on *initialization* and *termination conditions* that are specific to the actual HRL agent.

4.2.2. HRL properties: a gridworld example

A simple example will individuate the key properties of HRL and their differences with flat agents. A RL agent is placed in the simple gridworld of Figure 4.1. The primitive actions consist in moving to any of the four adjacent squares. The objective of the agent is to reach a goal position, randomly located in one of the four corners of the grid, upon which a positive reward is assigned. Even if the actual position of the goal is unknown, a flat RL controller can eventually discover the goal and learn an optimal policy for this task. Consider now a HRL agent with the four TEA "keep moving up", "keep moving down", "keep moving left" and "keep moving right". Each TEA moves the agent by reiterating the corresponding primitive action, and terminates if either an obstacle, the target, or a border is reached. The following observations can be made:

1. *TEAs allow state abstraction*. Consider again the gridworld of Figure 4.1, and assume the goal is in the upper-right corner. An optimal TEA sequence to solve this problem would be "keep moving right" then "keep moving up", regardless of the position of the agent, and of the size of the environment. If the agent is repositioned in a different room, in a different position, it can successfully reapply this policy again. The agent has therefore abstracted the specific environment, i.e., the gridworld of Figure 4.1, and learned a policy valid for similar ones, regardless of size and initial position. Conversely, a flat policy learned in this specific gridworld would not be transferable to another of different size.

2. *TEAs efficiently embed design knowledge in the learning process*. Assuming an even initialization of V, the initial behavior of a flat agent with a random or ϵ-greedy exploratory policy is a random walk, which can be quite suboptimal, depending on the size of the grid. By adopting the above TEAs, a designer can purposely avoid random walks. By doing so, the designer can use his or her own knowledge of the agent and of the environment, i.e., that the goal is

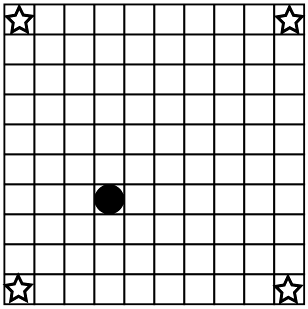


Figure 4.1: A simple gridworld where the agent (the circle) can move to an adjacent square as a primitive action. The four stars at the corner represent possible target positions.

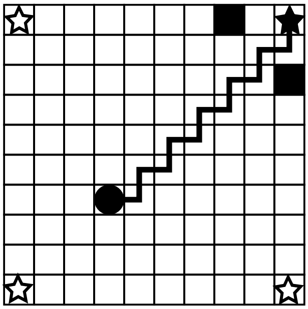


Figure 4.2: The policy of the flat agent reaches the target (black star) optimally. This or any other optimal policy cannot be discovered by the HRL agent due to the presence of obstacles (black squares).

in one of the corners, without explicitly assigning a policy to the agent, thus allowing it to learn, e.g., in which corner is the goal. Similarly to *optimistic initialization*, HRL and TEAs constitute a strategy to guide the learning process, rather than hard-coded solutions to tasks.

3. *TEAs constrain the set of discoverable policies*. Considering each applied primitive of an TEA separately as a single action, the HRL policy (which is a sequence of TEAs) can be transformed into one equivalent flat policy. It can be seen immediately that only a subset of all possible policies can be obtained with the given TEAs. For example, a flat policy as in Figure 4.2, where the agent alternatively moves right and up until it reaches the target position, cannot be described in terms of the "keep moving" TEAs, so it cannot be discovered by the HRL agent. This can potentially make it impossible to discover optimal policies, as is the case of Figure 4.2 after the introduction of obstacles in the grid. Conversely, all HRL equivalent flat policies can be discovered by the flat RL agent, as long as they involve the same primitives.

A further distinction is worth mentioning to fully understand the impact of TEAs on the learning process. RL is based on, but not limited to, the Markov decision process (MDP) formalism, for which the Markov property holds. Referring once more to the previous example, the value of a state (i.e., a position in the gridworld) with flat RL is function of the policy but not of previous history. The value of the same state when HRL is considered depends instead in whether the agent is currently executing a TEA, and furthermore on the specific TEA being performed. HRL policies are therefore non-Markovian, and TEAs are not compatible with MDPs. It is possible, however, to accommodate TEAs within a similar formalism, the Semi-Markov decision process (SMDP). Avoiding unnecessary details, SMDPs can be described as regular MDPs in which the time between one decision and the next is a random variable [104]. Opportune methodologies exist to extend RL learning algorithms, such as TD-methods, to SMDPs.

4.2.3. Restricted literature

Due to its generality, diverse methods of implementing TEAs are possible, and multiple instances of HRL exist in the literature. Nonetheless, Barto and Mahadevan [104] list three main approaches that will be here briefly summarized.

The *options* [105, 106] framework is the simplest implementation of the TEA concept. In this approach, the agent retains all primitive actions $u \in \mathcal{A}$, but has also access to a set of *options* o. These are defined as triplets $\langle \mathcal{I}, \pi_o, \beta \rangle$. $\mathcal{I} \subset \mathcal{S}$ represents those states where the option is available, i.e., the initialization conditions; $\pi_o(\boldsymbol{x}, u)$ is the fixed stochastic policy of the option giving the probability of action u in $\boldsymbol{x}$; $\beta(\boldsymbol{x})$ specifies the termination probability of o when in $\boldsymbol{x}$. The option framework represent a natural extension of classical RL, as any primitive action u can essentially be seen as a "one-step" option with $\mathcal{I} = \mathcal{S}$, $\pi_o(\boldsymbol{x}, u) = 1$ and $\beta(\boldsymbol{x}) = 1, \forall \boldsymbol{x}$. The most attractive feature of this HRL method is that, since the agent can directly execute primitives in addition to TEAs, the set of discoverable policies is the same one of flat RL. Although [105] introduced options as a-priori designed policies, further research [107–109] has proposed methods to discover, update or integrate the option set during learning.

Hierarchies of Abstract Machines (HAMs) [8] perform TEAs with the use of *machines*. Each contains a stochastic policy and a termination condition, similarly to options, from which they nonetheless differ significantly for three reasons. First, the agent initializes and switches between machines based on predefined fixed interconnections: the machines and their interconnections define the HAM. Second, the MDP-issued reward is used to update both in-machine policies and between-machine transitions. Third, the learning space of the agent is the cross-product of the MDP state space and the HAM state space, i.e., the agent policy depends also on the internal machine state. In practice, each machine is an a-priori designed combination of four possible machine states: *action* states execute a specific primitive action; *choice* states select the next machine state within a fixed selection; *call* states transition from the current machine to a different one; *stop* states interrupt the current machine and return control to the caller (see Figure 4.3). The HAMs architecture offers a tradeoff between embedded knowledge, obtained by appropriately designing the machines, and autonomy of learning. It can be demonstrated that, when considering the augmented learning space, classical RL learning schemes such as Q-learning can be applied to HAMs with a few modifications.

MAXQ [110] is similar to HAMs in that the learning space of the agent is a-priori constrained into a graph of *subtasks*, whose structure is defined by the designer. Starting from the *root*, each subtask calls one of its own *children* subtasks according to its policy, until a primitive (which are "one-step" special subtasks) is performed and reward is obtained. Subtasks are similar to options in that they also have initialization and termination conditions. The total discounted reward observed by children during their execution is then also transferred to the parent. This, together with a *completion function*, is used to update the value of the parent, in a recursive manner. Two observation can be made. First, by adding the exact "stack" of tasks and subtasks being executed to the MDP state, the resulting learning process is Markovian. Second, MAXQ converges to a hierarchical policy that is *recursively*

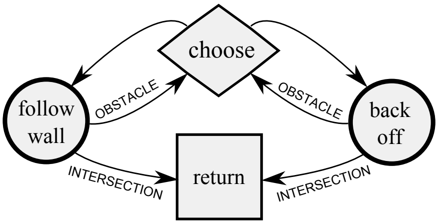


Figure 4.3: Example of machine architecture (adapted from [8]) for obstacle negotiation. The round states are action states, the square is a stop state, and the diamond is a choice state.

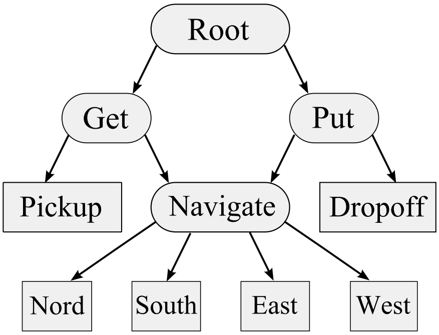


Figure 4.4: The MAXQ architecture for the taxi problem (adapted from [110]). With the exception of Root, subtasks can either execute a primitive or call one of their own subtasks. The Navigate subtask is actually shared between Get and Put.

optimal rather than hierarchically optimal, i.e., each subtask's policy is individually optimal, but the equivalent flat-policy is not optimal with respect to the set of discoverable policies. Third, internal pseudo-rewards can be added to the "natural" reward, but exclusively while learning the subtasks, in order to accelerate and guide this process. As an example of MAXQ, [110] proposes the taxi problem (see Figure 4.4), where learning the *Navigate* subtask early on is beneficial for learning the parent subtasks as well.

Feudal Reinforcement Learning [111] represents an extreme approach to hierarchy, similar to *layered control* [112]. The agent is split in a top-down tree of *managers*, rather than in a graph of subtasks or in a hierarchy of machines. The higher the position of the manager, the more high level the domain of its policy, and the more coarse its own state representation. With the exception of the "root" manager at the first level and the bottom managers at the final level, each manager has exactly one superior but possibly several subordinates. Similarly to MAXQ, the superior/subordinate connections are fixed by the designer. Superiors assign goals to their subordinates, and assign them reward when and if the goal they assigned them is reached. Feudal RL presents two important aspects. First, goals and rewards are not shared between managers, so that an efficient subordinate will receive positive reward from an inefficient superior assigning an incorrect goal. Second, contrarily to HAMs and MAXQ, in Feudal RL only the lowest level managers are allowed to execute primitives (Figure 4.5).

4.2.4. Safe Hierarchical Reinforcement Learning

This section investigates the application of HRL methods to Safe Reinforcement Learning (SRL) [13], which constitutes the novel field of SHRL.

Policies learned via *State abstraction* are valid for different, similar environments. Furthermore, abstraction makes the use of graph methods easier, as it reduces

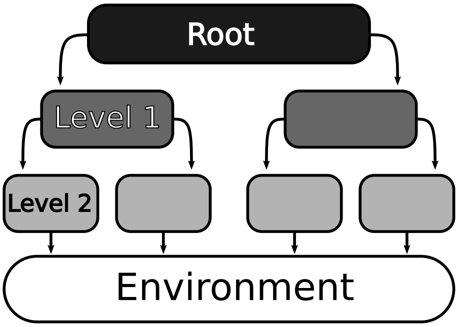


Figure 4.5: An example of managerial architecture for Feudal RL with three levels. Each manager can assign a task only to one of its subordinates, but only lowest-level managers (level 2) can actually perform primitives.

the size of the learning space, lowering the necessary refinement of tilings, and simplifying the definition of OEs. Therefore, state abstraction contributes to solving the *challenge of robustness*.

There are diverse ways in which abstraction can be obtained. Consider for example a classic sequential task as the one described in [113]. An agent is in a dark, locked room. The room contains a glowing switch to turn on and off the lights, and a second switch that opens the door to a second room, which contains a "charger" state in which a specific charging action is positively reinforced. Before the agent can charge itself, a number of intermediate sub-goals must be reached as well. Certain state features are fundamental to reaching a specific subgoal (e.g., the light must be on for the second switch to be visible), but irrelevant for other subgoals (e.g., in the charger state it is irrelevant whether the light is on or not). Similarly, in the taxi problem of Figure 4.4, whether or not the taxi is carrying a passenger is irrelevant to the agent for the sake of learning how to navigate from one state to the other. If an HRL agent can abstract, i.e., generalize between states that share relevance for the task, the complexity and dimensionality of the learning space is effectively reduced. Additionally, if an option or subtask reduces the learning space, those dynamics that involve only irrelevant states for a subtask can be temporarily ignored for the sake of learning that specific subtask. This is in the spirit of the "recursive optimality" of [110]. In the room environment, the agent can "forget" its knowledge about turning on lights or pushing buttons as soon as it opens the door. Similarly, in the MAXQ taxi problem, the agent does not need to know hot to pick up a passenger while it navigates. In conclusion, this allows to obtain more flexible policies and to generate lower-dimensional, more efficient graphs, with a smaller subset $\mathcal{A}_{sub}$ and a more refined tiling, which reduces the added uncertainty during propagation.

Embedding design knowledge and *constraining the set of discoverable policies* are beneficial for the safety of exploration. Design knowledge can introduce reactivity towards unsafe states. Consider for example the obstacle-negotiating set of machines of Figure 4.3: the detection of an obstacle triggers a choice state into avoiding or circumnavigating the obstacle. Even if this cannot guarantee that the obstacle will be avoided, it directs the agent toward learning a safe policy. By appropriately assigning initialization and termination conditions, the same guidance can be achieved with options, MAXQ and other HRL methods.

Constraining the set of discoverable policies can be an inconvenience of adopting HRL instead of a flat RL, since the optimal policy might not be among the set. However, this constraint could in principle remove the unsafe policies from the discoverable set. As an example of how this can be achieved, consider a Feudal RL architecture, where only lowest-level managers are allowed to perform primitive actions. In this case, it is sufficient to guarantee that the lowest-level managers' policies are safe, regardless of the goal assigned by superiors, to guarantee that the hierarchical policy is also safe. In practice, enforcing constraints that are favorable to safety highly depends on the specific task.

Given the above considerations, HRL is shown to be beneficial to safety, in addition to address the *challenge of robustness* as well as reducing the curse of dimensionality and accelerating learning. In the following chapter, a specific SHRL method denominated *Virtual Safety Training* (VST) is presented.

4.3. Virtual Safety Training

This section presents the VST strategy, which under given assumptions can provide safety of exploration. When implementing VST there is no specific restriction in terms of how hierarchy is structured, and in how TEAs are obtained. For the sake of this chapter, machines, options, subtasks and any other similar structure are referred to with the acronym *MOS*.

Consider a continuous bijective function $\mu : \mathcal{S} \to \prod^{\times}_{j\in\{1,\dots,m\}} \mathcal{P}^j$ which assigns to each $\boldsymbol{x} \in \mathcal{S}$ a unique set of vectors $\{\boldsymbol{p}^1, \dots, \boldsymbol{p}^m\}$. Given the dynamics[1] of $\boldsymbol{x}$ and of the vectors $\boldsymbol{p}^i$, and given $\mu_i : \mathcal{S} \to \mathcal{P}^i$, it holds that

$$\dot{\boldsymbol{p}^i} = \frac{\mathrm{d}\mu_i(\boldsymbol{x})}{\mathrm{d}t} = \frac{\mathrm{d}\mu_i(\boldsymbol{x})}{\mathrm{d}\boldsymbol{x}} \cdot \dot{\boldsymbol{x}} = \frac{\mathrm{d}\mu_i(\boldsymbol{x})}{\mathrm{d}\boldsymbol{x}} \cdot \mathcal{D}(\mu^{-1}(\boldsymbol{p}^1, \dots, \boldsymbol{p}^m), u) = \mathcal{D}^i(\boldsymbol{p}^1, \dots, \boldsymbol{p}^m, u), \quad (4.1)$$

and given components $(p^i_1, \dots, p^i_{k_i})$ of $\boldsymbol{p}^i$, the notation

$$\dot{p}^i_a = \mathcal{D}^i_a(\boldsymbol{p}^1, \dots, \boldsymbol{p}^m, u), \quad (4.2)$$

is used to indicate the derivative of component p^i_a, $a \in \{1, \dots, k_i\}$. A function μ such that the above holds will be referred to as a *state projection*, of which $\boldsymbol{p}^i$ are the *projected states*. Figure 4.6 shows an example of a state projection. The concepts of safe state space, fatal state space, lead-to-fatal states are still valid in the projected space. Specifically, given a projected state $\boldsymbol{p}^i$, it is possible to identify the projected safe state space PSSS_i

[1] For ease of exposition, the state is assumed to be continuous: $\dot{\boldsymbol{x}} = \mathcal{D}(\boldsymbol{x}, u)$. However, the definitions and results of this section can be extended to discrete state spaces as well.

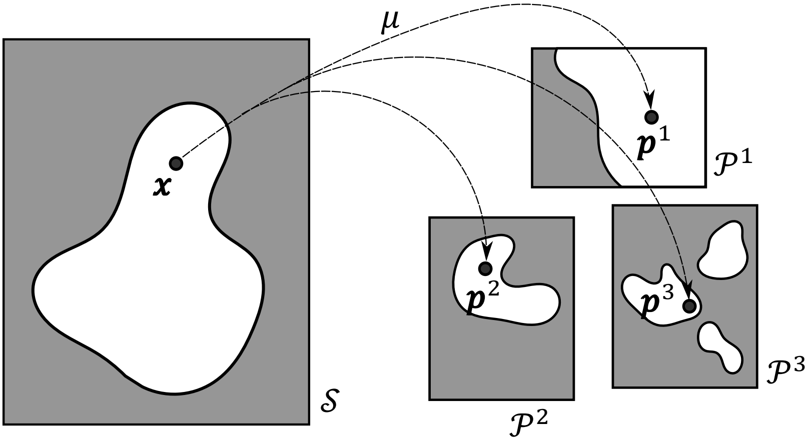


Figure 4.6: Example of projection function μ, which assigns to each state $\boldsymbol{x}$ a tuple $\{\boldsymbol{p}^1,\boldsymbol{p}^2,\boldsymbol{p}^3\}$. Note that the FSS, represented as the gray subset of $\mathcal{S}$, has an equivalent in the three PFSS_i of the projected spaces $\mathcal{P}^i$.

$$\mathrm{PSSS}_i(\boldsymbol{p}^1,\dots,\boldsymbol{p}^{i-1},\boldsymbol{p}^{i+1},\dots,\boldsymbol{p}^m) = \{\boldsymbol{p}^i|\mu^{-1}(\boldsymbol{p}) \in \mathrm{SSS}\} \tag{4.3}$$

which represents the safe values of $\boldsymbol{p}^i$ given the value of the remaining projected states. The projected fatal state space PFSS_i is defined analogously. Recalling the Definition 5 of safe control

$$\boldsymbol{\sigma}(\boldsymbol{x},u,t) \in \mathrm{SSS},\ \forall t \in [t_1,t_2], \tag{4.4}$$

it is $\boldsymbol{\sigma}(\boldsymbol{x},u,t) = \boldsymbol{\sigma}(\mu^{-1}(\boldsymbol{p}^1,\dots,\boldsymbol{p}^m),u,t)$ and

$$\boldsymbol{\sigma} \in \mathrm{SSS} \Leftrightarrow \boldsymbol{p}^i(t) \in \mathrm{PSSS}_i(\boldsymbol{p}^1(t),\dots,\boldsymbol{p}^{i-1}(t),\boldsymbol{p}^{i+1}(t),\dots,\boldsymbol{p}^m(t)). \tag{4.5}$$

It is therefore possible to rewrite the safety problem in terms of one arbitrary $\boldsymbol{p}^i$. However, some complications occur in this formulation. First, the SSS and therefore PSSS_i are unknown. Then, if $\dot{\boldsymbol{p}}^j \neq 0$, PSSS_i might vary with time. Apparently, projecting the state has the only effect of complicating the problem. However, specific hierarchical structures can be used to exploit systems that accommodate certain assumptions, as described in the remainder of this section.

It was observed in the previous section how HRL can reduce the dimensionality of the learning space. Consider once again the taxi problem example and assume knowledge about relevant and irrelevant features is embedded in the HRL agent structure. When the taxi is navigating, the position and destination of the passenger has no influence on how the taxi moves. The state can be separated as

$$\mu(\boldsymbol{x}) = \{\boldsymbol{p}^1, \boldsymbol{p}^2, \boldsymbol{p}^3\} = \{\mathtt{taxi}, \mathtt{dest}, \mathtt{pass}\}, \tag{4.6}$$

where `taxi` indicates the position of the taxi, and `pass` and `dest` indicate the position of the passenger and his destination. It is evident that $\frac{\partial \mathcal{D}^1(\mu(\boldsymbol{x}),u)}{\partial \boldsymbol{p}^2} = \frac{\partial \mathcal{D}^1(\mu(\boldsymbol{x}),u)}{\partial \boldsymbol{p}^3} = 0$. The following definition generalizes the concept.

Definition 11 *Let* $\mu : \mathcal{S} \to \prod^{\times}_{j\in\{1,\dots,m\}} \mathcal{P}^j$ *be a state projection, and let* $\boldsymbol{p}^i = (p^i_1,\dots,p^i_{k_i}) \in \mathcal{P}^i,\boldsymbol{p}^j = (p^j_1,\dots,p^j_{k_j}) \in \mathcal{P}^j$*,* $i,j \in \{1,\dots,m\}$ *be two projected states. Then, component* p^i_a*,* $a \in \{1,\dots,k_i\}$ *is* ***independent*** *of* $\boldsymbol{p}^j$ *if* $\forall b \in \{1,\dots,k_j\}$*, it is* $\frac{\partial \mathcal{D}^i_a}{\partial p^j_b} = 0$*. If,* $\forall a \in \{1,\dots,k_i\}$*,* p^i_a *is independent of* $\boldsymbol{p}^j$*,* $\boldsymbol{p}^i$ *is independent of* $\boldsymbol{p}^j$*.*

If $\boldsymbol{p}^i$ is independent of all $\boldsymbol{p}^j$, then transitions $\{\boldsymbol{p}^i(k), u, r, \boldsymbol{p}^i(k+1)\}$ can be effectively modeled as an MDP. In HRL terms, projecting the state in such a way allows to design a MOS whose initialization conditions, policy π and termination conditions are defined within $\mathcal{P}^i$ alone.

A second notable simplification can be found by using a relative state representation, e.g., $\mu(\boldsymbol{x}) = \{\boldsymbol{p}^1,\boldsymbol{p}^2\} = \{\boldsymbol{x}_0, \boldsymbol{x}_{\text{rel}}\}$, $\boldsymbol{x} = \boldsymbol{x}_0 + \boldsymbol{x}_{\text{rel}}$, for which it holds $\frac{\mathrm{d}\mathcal{D}^1(\mu(\boldsymbol{x}),u)}{\mathrm{d}t} = 0$. Again, this can be generalized as the following.

Definition 12 *Let* $\mu : \mathcal{S} \to \prod^{\times}_{j\in\{1,\dots,m\}} \mathcal{P}^j$ *be a state projection, and let* $\boldsymbol{p}^i = (p^i_1,\dots,p^i_{k_i}) \in \mathcal{P}^i,\boldsymbol{p}^j = (p^j_1,\dots,p^j_{k_j}) \in \mathcal{P}^j$*,* $i,j \in \{1,\dots,m\}$ *be two projected states. Then,* $\boldsymbol{p}^i$ *is* ***relative*** *to* $\boldsymbol{p}^j$ *if* $\forall b \in \{1,\dots,k_j\}$*, it is* $\frac{\mathrm{d}\mathcal{D}^j_b}{\mathrm{d}t} = 0$*.*

A relative representation can be extremely useful for an HRL design. One advantage is that the dynamics can be linearized or otherwise approximated around a nearby state $\boldsymbol{x}_0$. $\mathcal{D}^i$ can therefore have a much simpler structure than $\mathcal{D}$. Furthermore, when considering risk perception, the agent is likely to have information about the safety of its surrounding. In terms of an HRL implementation, such a projection can be used to devise a MOS whose initialization conditions, policy π and termination conditions are a function of $\boldsymbol{p}^2 = \boldsymbol{x}_{\text{rel}}$ and of the value at the instant of projection of $\boldsymbol{p}^1 = \boldsymbol{x}_0$, since the reference value is constant during the execution of π until termination. It is assumed that the following assumption holds, such that $\dot{\boldsymbol{p}}^i = \mathcal{D}^i(\boldsymbol{p}^i,\boldsymbol{p}^j_0,u)$.

Assumption 5 *There is at least one projection* μ *of* $\mathcal{S}$*, with at least one projected state* $\boldsymbol{p}^i$ *such that, given any other projected state* $\boldsymbol{p}^j$ *with* $j \neq i$ *and* $j \in \{1,\dots,m\}$*,* $\boldsymbol{p}^i$ *is independent of* $\boldsymbol{p}^j$*, relative to* $\boldsymbol{p}^j$*, or both.*

Chapter 2 introduced the concept of boundedness of an uncertain model $\hat{\mathcal{D}}$ with respect to a "crisp" model $\mathcal{D}$. It was shown how this can be the result of uncertainty in the identification of the model parameters, or can derive from the estimating the deterioration of a vehicle whose model was previously identified. Analogously, the following is assumed:

Assumption 6 *Let μ be a state projection with a projected state $\boldsymbol{p}^i$ for which Assumption 5 holds. Then $\exists$ a function $\hat{\mathcal{D}}^i \Rightarrow \forall \boldsymbol{x} \in \mathcal{S}, \forall u \in \mathcal{A}$, $\mathcal{D}^i(\boldsymbol{p}^i, \boldsymbol{p}_0, u) \in \hat{\mathcal{D}}^i(\boldsymbol{p}^i, \boldsymbol{p}_0, u)$.*

Combining Assumption 5 and Assumption 6 means that the agent can estimate the transitions of $\boldsymbol{p}^i$ according to a bounding model $\mathcal{D}^i$. However, PFSS_i and the reward function $\mathcal{R}$ might not depend uniquely on $\boldsymbol{p}^i$.

In terms of safe exploration, it would be appealing to apply a "safety filter", such as SHERPA, to the projected space $\mathcal{P}^i$ so that $\boldsymbol{p}^i(t) \in \mathrm{PSSS}_i$, $\forall t \in [t, \infty]$. However, the problem introduced and discussed in Chapter 2 assumes a static FSS, which guarantees the validity of backups. If PFSS_i is a function of other projected states $\boldsymbol{p}^j$, it can change during the execution of the MOS. This increases the complexity of applying a SHERPA-like filter. VST is attempted instead.

The strategy of VST can be defined as learning a "virtual" policy π_{V} off-line such that

$$\boldsymbol{\sigma}(\boldsymbol{p}^i, \pi_{\mathrm{V}}(\boldsymbol{p}^i, \mathrm{PSSS}_{\mathrm{V}}), t) \in \mathrm{PSSS}_{\mathrm{V}}, \forall t \in [t_1, t_2], \forall \mathrm{PSSS}_{\mathrm{V}} \in \mathcal{B}, \tag{4.7}$$

where $\mathrm{PSSS}_{\mathrm{V}}$ is the *belief* of the agent with respect to the projected safe state space, and $\mathcal{B}$ is the belief set. For each $\mathrm{PSSS}_{\mathrm{V}}$, policy π_{V} is learned off-line with the dynamics indicated by the bounding model, assigning negative reward when a fatal transition is encountered.

In the most generic case, where PSSS_i is an unknown, time-varying set, defining an *exhaustive* set $\mathcal{B}$, i.e., a set for which

$$\forall t, \exists \mathrm{PSSS}_{\mathrm{V}} \in \mathcal{B} \Rightarrow \mathrm{PSSS}_i(t) = \mathrm{PSSS}_{\mathrm{V}}, \tag{4.8}$$

is likely to be intractable. However, some simplifications can be applied. First of all, if SSS can be estimated (e.g., via risk perception), this restricts the set $\mathcal{B}$ to those compatible with this estimate. Additionally, if $\boldsymbol{p}^i$ is *relative*, then PSSS_i is constant during each execution of the policy, so that $\mathcal{B}$ is not time-varying. A further simplification comes from applying state abstraction to PFSS_i. Multiple states $\boldsymbol{p}^i \in \mathcal{P}^i$ can be aggregated, so that PFSS_i can be approximated as a finite number of fatal aggregated states.

Training is then performed in this "virtual", projected, abstracted representation of the state space. The better the belief set $\mathcal{B}$ can approximate the PSSS_i, the safer and more efficient the learned policy. In the event that the belief set is exhaustive, π_{V} can guarantee safety.

Regardless of the completeness of $\mathcal{B}$ and of the efficiency in approximating PSSS_i, the agent must convert its current information on the environment, e.g., deriving from its risk perception, into the most similar belief among $\mathcal{B}$ and subsequently apply the corresponding virtual policy π_{V}.

A MOS policy obtained and designed as described above is oriented toward safe exploration. However, if a MOS whose policy is not learned through VST executes a primitive, the aforementioned benefit is lost. In order to avoid this occurrence, the discoverable policies should be constrained in such a way that only VST-trained MOS are allowed to perform primitives, at least while safe exploration is in progress.

This section has formalized the VST strategy, which is part of SHRL. Section 4.4 and Section 4.5 present two implementations of the strategy as two case studies for the method. Specifically, in Section 4.4 VST is applied to a discrete problem, with a deterministic transition model. In this case, an exhaustive belief set is created such that, with the obtained policy π_{V}, exploration is entirely safe. In Section 4.5, VST is applied instead to a continuous application, where the UAV model is uncertain. In this case, a more rough abstraction is performed to obtain a non-exhaustive belief set $\mathcal{B}$.

4.4. VST with deterministic dynamics and exhaustive belief set

In this task, the agent navigates a ground robot in an unknown maze, which is surrounded by walls and presents obstacles, which are a source of possible collisions and which can be detected at short range. The maze contains goal states, whose location is unknown, and which the agent must find.

VST is applied in order to avoid collisions. In particular, the HRL structure used here is inspired by the *layered control* philosophy of Brooks [112] and by the top-down hierarchy of Feudal RL [111]. A state projection is applied for which a relative and independent projected state exists, and a highly abstracted "lowest-level" MOS is defined on that projected state. The simplicity of dynamics and the discreteness of the environment allow an exhaustive belief set, so that the policy π_{V} obtained through VST might be sub-optimal, but is safe. Additionally, VST prevents blind-search, and π_{V} performs better than a simple, hard-coded obstacle evading policy.

4.4.1. Introduction to maze navigation and mapping

Indoor navigation is a field where both autonomous [114] and semi-autonomous [115] platforms can benefit from the application of HRL. In particular, maze navigation exemplifies the advantages of abstraction during learning. Applying classical, "flat" RL algorithms to mazes usually implies very long learning times, due to the typically large state space of these tasks, and due to the difficulty of implementing appropriate function approximators. However, the problem can be simplified by abstraction. For example, an agent that has learned how to avoid one obstacle could reapply this knowledge to avoid similar ones. Higher levels of abstractions can be devised; e.g., if the agent has learned what constitutes a wall, it can learn wall-following behavior, and remember which walls have been previously followed. These and other relatively simple abstractions can be used to speed up learning; however, the corresponding behavior very often originates from the designer itself. For example, to implement wall or corridor following behaviors, the designer needs to properly define which actions are more appropriate and devise an appropriate reward. In this section, this behavior is discovered autonomously by the agent.

In addition to VST, the proposed HRL agent utilizes maps in order to decide where to navigate. Mapping means using sensor readings to generate an internal representation of the environment. Typically, the agent starts exploring with a small map of its immediate surroundings or with no map at all, but expands on this as more measurements are performed. Two main representations for maps exist. As the name suggests, *grid-based* maps partition the environment into an evenly spaced grid. In theory, each cell can be observed or unobserved, and observed cells can be either empty or occupied. However, in order to cope with reading errors, occupancy of a cell is often treated as a continuous value, e.g., as with *evidence grids* [116]. If the value of occupancy of a cell is below a set threshold it is considered empty, otherwise it is treated as full. Grid-based maps constitute a quantitative representation for an environment, while *topological* maps are a qualitative one. These maps do not use grids but graphs: vertices represent locations, such as rooms, corridors, or landmarks; edges indicate which locations are reachable from one another. Hybrid representations known as "grid-topological" [117] have been attempted as well in the hope of combining the simple interpretation and use of topological maps with the more precise nature of grids.

4.4.2. Abstraction and training description

This section explains the proposed approach. After briefly describing the agent, the two main elements of the approach are illustrated: the hierarchical control for local navigation, and the global exploration strategy, based on mapping and RL.

Agent description

The agent considered in this chapter represents a ground robot equipped with distance sensors. The robot is assumed to be able to travel at constant speed in the direction it is facing, to brake with negligible braking distance, and to turn on the spot. The discrete primitive actions of the agent are advancing one square, turning $\frac{\pi}{2}$ (clockwise), or $-\frac{\pi}{2}$ (counterclockwise). The agent navigates with the aid of a map, not provided beforehand but built online based on sensor readings. Three close range distance sensors (e.g., sonars) are positioned at the front, right and left of the robot, detecting the nearest obstacle within three squares of distance in a straight line (see Figure 4.7). A compass detects which of the four cardinal directions the agent is facing at each time. Assuming perfect readings of the sensors, a map is built progressively which is consistent with the environment. It is assumed that the robot performs perfect odometry, so that the agent knows its position within the map at all times.

State projection

The original state $\boldsymbol{x}$ of the agent in the flat MDP is given by the absolute position x_1, x_2 and by the heading $\chi \in \{-\pi, -\frac{\pi}{2}, 0, \frac{\pi}{2}\}$ of the robot. Even though the state is discrete, the learning space can grow very fast by simply enlarging the maze. In terms of safety, the FSS represents all positions x_1, x_2 that are occupied by a wall or an obstacle. Technically, the agent sensors are sufficient to avoid obstacles if an hard-coded rule is applied, e.g., "when facing an obstacle, turn right". This limitation could be inserted in a MOS, e.g., in a machine. Using this or other simple responses can prevent collisions but can also cause trajectory loops or similar other complications that are difficult to prevent if the actual environment is not available. Therefore, the resulting policy might be safe, but at the cost of high initial

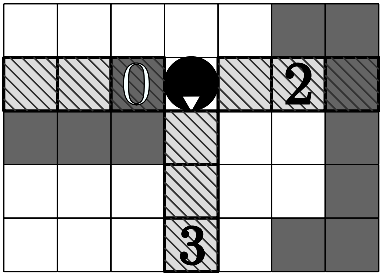


Figure 4.7: An example of sensor reading. The circle represents the robot, the triangle its heading, and the gray squares the obstacles. Sensors are positioned at the front, right, and left of the agent, and detect obstacles in a straight line with a range of up to three squares.

inefficiencies.

In order to apply HRL and specifically VST, the following state projection is applied. First of all, the maze is subdivided into groups of 3-by-3 squares, called *tiles*. Consider the tile containing the square currently occupied by the robot. Assigning an arbitrary orientation χ^{tile} to the tile, the position of the robot can be indicated as its position p within the tile, and the relative orientation χ_{rel} of the robot with respect to that of the tile. Therefore

$$\{\text{tile}, p, \chi_{\text{rel}}\} = \mu^{\chi^{\text{tile}}}(\boldsymbol{x}) \tag{4.9}$$

is a bijective function, given a unique choice of tiles and an arbitrary orientation χ^{tile}. The projected state p is independent of and relative to tile, since the robot dynamics are the same in all tiles. The first goal of the HRL design will be to perform VST appropriately by learning a policy that, while within a tile, will avoid collisions. This policy will then be inserted within a MOS in order to guarantee a safe exploration.

Virtual Safety Training for local navigation

Once the tile state projection is applied, the agent learns a policy in the corresponding projected virtual environment of Figure 4.8. This *minigrid* consists in a grid $\mathcal{P}$ of 3-by-3 *cells*, plus an additional row of goal states $\mathcal{P}^{\text{goal}}$. The objective of the agent is to learn how to reach these states in the presence of obstacles. Therefore, the orientation of the tile (which corresponds to the first three rows of the minigrid) is the same as the direction to goal of the agent.

State abstraction reduces the learning space of the virtual policy. In this application, the original agent can be abstracted into the *trainee* of Figure 4.8. The trainee agent does not have an orientation, and can move in any of the four adjacent cells. Therefore, the state of the trainee is only its position p. During training, the trainee is assumed to know the disposition of the obstacles inside the minigrid. At least one of the cells must contain the trainee, so that a total of $2^{12} - 2^3 = 4088$ obstacle placements $\mathcal{O}$ are possible. Since fatal occurrences consist in colliding with obstacles, obstacle placements form also the belief set $\mathcal{B}$, which is an exhaustive representation of PFSS.

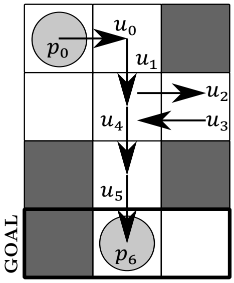


Figure 4.8: In this sample episode, the environment contains four occupied squares, included a goal state. The agent starts in p_0 and descends towards goal p_6 in six steps. As the agent performs a deviation with action u_2, the obtained trajectory is suboptimal.

The training is divided into episodes, at the start of which the trainee is positioned in a random, empty cell of $\mathcal{P}$, and a compatible disposition of $\mathcal{O}$ is selected. The trainee can attempt moving to any adjacent cell by conventional actions $\mathcal{A}^{\text{train}} = \{\texttt{up}, \texttt{down}, \texttt{right}, \texttt{left}\}$. If the cell is empty, the trainee moves with probability one and receives a reward of -1. Otherwise, the action is discarded for this iteration. The trainee chooses a random action with probability 0.1, and otherwise takes the action that, during the episode, has been selected the least amount of times from its current position. An episode terminates as soon as the trainee reaches a goal position $p_g \in \mathcal{P}^{\text{goal}}$, or if the 100^{th} iteration is reached, including discarded actions.

Given k the number of moves in an episode, the trainee, initially in p_0, performs k trajectories $\{\mathcal{O}, p_i, u_{i:k-1}, R, p_k\}$, $i < k$. Each trajectory is a motion pattern starting from position p_i at timestep i, all terminating in p_k at time k, after sequence of actions $u_{i:k-1}$. Each trajectory yields an undiscounted return $R{=}i{-}k$. If the episode terminates before the 100^{th} iteration, p_k is always one of the goals p_g. The trainee can then use each trajectory as a TEA when in position p_i to get to that goal, given those same obstacles, with a return R. The same motion is also reused whenever possible by applying appropriate transformations to the environment, e.g., by mirroring the environment.

In case the iteration limit of 100 is reached, it is assumed that the goals can not be physically reached, given obstacles $\mathcal{O}$, from all cells visited during the episode. These trajectories are also used to teach the trainee, with the following modifications. First, R is replaced by -100 for all trajectories, as an indication that they all fail in reaching the goals. Second, each trajectory is replaced by three trajectories with $p_k \in \mathcal{P}^{\text{goal}} = \{p_g^1, p_g^2, p_g^3\}$. Unsuccessful trajectories are then in the form: $\{\mathcal{O}, p_i, u_{i:k-1}, -100, p_g^1\}$, $\{\mathcal{O}, p_i, u_{i:k-1}, -100, p_g^2\}$ and $\{\mathcal{O}, p_i, u_{i:k-1}, -100, p_g^3\}$.

Valid trajectories are stored in a table $\mathtt{T} : \mathcal{O} \times \mathcal{P} \times \mathcal{P}^{\text{goal}} \times \mathcal{A}^{\text{train}} \to \mathbb{R}$ that maps the combination of environment, starting and ending positions, and action of each trajectory to the return R. The $4088 \cdot 9 \cdot 3 \cdot 4$ entries of $\mathtt{T}$ are initially empty. At the end of an episode, and once for each trajectory, the content of the corresponding entry in $\mathtt{T}$ is compared to the new return R: if the entry is empty or has a lower value, it is replaced by R. After convergence, $\mathtt{T}$ indicates if a goal can be reached

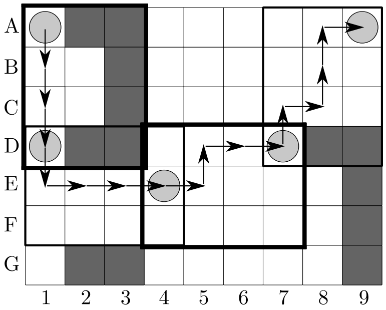


Figure 4.9: Sample of maze navigation obtained through iteratively adopting the minigrid logic of T.

for given environment and position, and the number of moves necessary to do so when starting with action u_i.

Square navigation

The table T can then be used to navigate the agent within the maze, provided that:

- the agent has a belief of the surrounding obstacles;
- a direction to goal, coinciding with χ^{tile}, is assigned.

Navigation is performed as follows. The agent places upon the map a minigrid formed by the current tile and the three neighboring squares in the given direction to goal. Each cells of the minigrid corresponds to a square of the maze. Setting aside for the moment the presence of unobserved squares, assume that the agent has a map providing an exact disposition of the obstacles, and therefore the necessary belief for the agent. Then, according to the current position p_i inside the minigrid, T is consulted to obtain the optimal trainee action up, down, etc., which is then "decomposed" into the equivalent primitive actions. Figure 4.9 illustrates an example of navigation from state A1 to state A9. The agent is initially in A1, and is instructed to go South. It positions the minigrid between A1 and D3, and reaches goal D1. Then, it is commanded to proceed East, so it positions the minigrid in D1-F4. This time multiple goals are available, and the agent ends in E4. Following the same identical steps, it then continues East through D4-F7, and finally North through A7-D9.

As it can be seen, shortest-path maze navigation can be obtained by iterating the policy contained in T. However, determining which direction to goal is optimal is not trivial. For example, the direct route between A1-A9 in figure Figure 4.9 is blocked by an obstacle. The agent must understand the presence of the obstacle, and then circumnavigate it, in order to avoid unnecessary or even unsafe actions (e.g., collisions).

Tile navigation

Obstacle circumvention can be achieved by further abstracting the state of the agent. The policy of T indicates which actions to apply in order to reach the goal

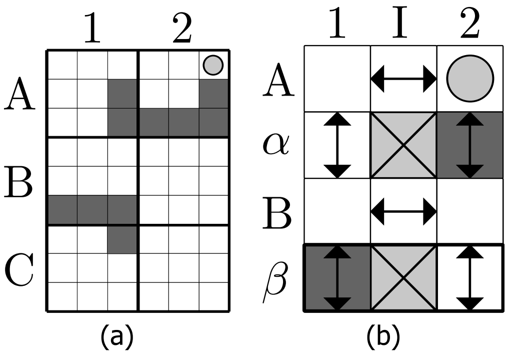


Figure 4.10: Transitions between tiles (a) are abstracted into a macrogrid (b), which allows to circumvent obstacles.

cells in a given direction, and takes into account the presence of obstacles. In principle, the same policy can be used for tiles as well, by replacing each primitive action, e.g., `North`, with a TEA, e.g., "move one tile to the North"; this TEA can be performed again with `T` applied to squares. In this way, tiles are abstracted into cells of a *macrogrid*. However, one final consideration is needed in how to abstract the presence of obstacles within the macrogrid representation.

An example will clarify how to do so. Figure 4.10a shows a portion of map and its six constituent tiles A1, A2, B1, and so on. The macrogrid consists in a 4-by-3 grid of cells, of which two types exist. *Position* cells A1, A2, B1 and B2 (Figure 4.10b) represent the tiles with the same name. If the agent is in a tile, it is considered to be inside the corresponding position cell. The remaining cells (with the exception of αI and βI) are *transition* cells. If a transition cell is occupied, it is not possible to move between the two adjacent position cells, i.e., the two adjacent tiles. For example, in Figure 4.10, obstacles prevent paths A2-B2 and B1-C1, so $\alpha 2$ and $\beta 1$ are occupied. The occupancy of transition cells can be assessed by the agent by positioning a minigrid on the map, so that the first three rows overlap one of the two tiles, and then checking `T` to see if any goal can be reached. Finally, cells αI and βI represent invalid transitions, and as such are always considered occupied. In this way, the macrogrid cells are abstraction of the tiles, as well as of transitions between tiles. The policy of `T`, defined for the microgrid, is also valid for the macrogrid, taking into consideration the different transition and occupancy rules of cells. Therefore, `T` can be used to navigate the robot between tiles as well.

Figure 4.11 shows an application of the combined hierarchical control when the map contains unobserved squares. A grid of 3-by-2 tiles is then placed on the map, so that the goal tiles are located in the direction to goal, which in this example is East. The agent is initially in the top-right corner of tile A2. In order to promote exploration, all transition cells of the macrogrid are initially considered to be empty. Transitions between tiles are indicated by applying `T` to the macrogrid. Then, a minigrid is positioned as to include the current tile and the three neighboring squares of the destination tile, which represent the goal cells. Table `T` is then

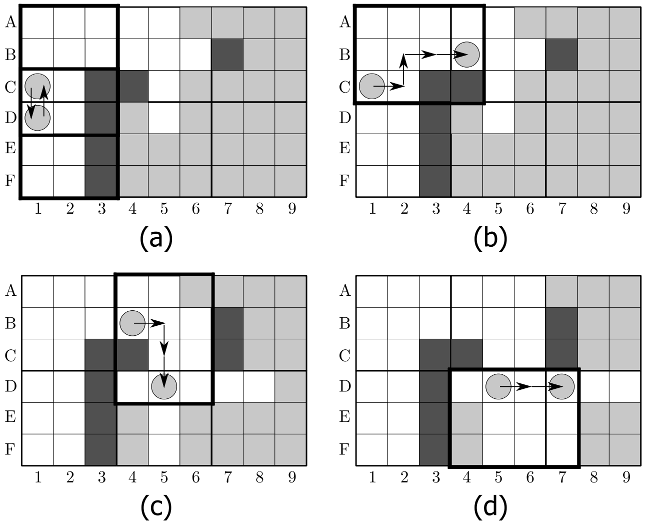


Figure 4.11: In (a), the agent receives instructions South, East, East from the macrogrid MOS, to which it cannot abide due to the presence of obstacles. A new path (North, East, East) is then issued and executed, until in (b) the microgrid MOS indicates that the given instruction might lead to a collision with unobserved obstacles. A third and final path is obtained by the macrogrid (South, East) which is then carried on with success in (c) and (d).

consulted again to move between squares. Any unobserved square of the map is represented inside the minigrid as an occupied cell, in order to prevent collisions with unobserved obstacles. It can be that, due to the presence of occupied or unobserved squares, the tile transition indicated by the macrogrid cannot be completed. The corresponding transition cell is marked as occupied, and the updated macrogrid is consulted again to see if an alternative path is available. For example, in Figure 4.11a, the initial suggested tile transitions to move East are South, East, East. The South transition is immediate. However, when placing the minigrid for the following move to East, the corresponding entry of T returns -100, meaning that the tile transition is impossible. The corresponding transition cell is updated to occupied, and the agent reroutes as North, East, East. The robot goes up one tile to the original position (North), and then one tile right (East) in Figure 4.11b. The final tile transition East is again denied, since the unobserved squares might contain obstacles. The macrogrid then indicates South and East, which the agent executes in Figure 4.11c and in Figure 4.11d.

Hierarchical policy structure

In terms of HRL nomenclature, "moving between tiles" and "moving between squares" are two different MOS policies, combined in the feudal structure of Figure 4.5. In order to navigate, each MOS needs an input from a superior assigning the direction of motion, and a belief concerning the disposition of obstacles. Note that the two beliefs utilized for tile and for square navigation are different: the macrogrid belief

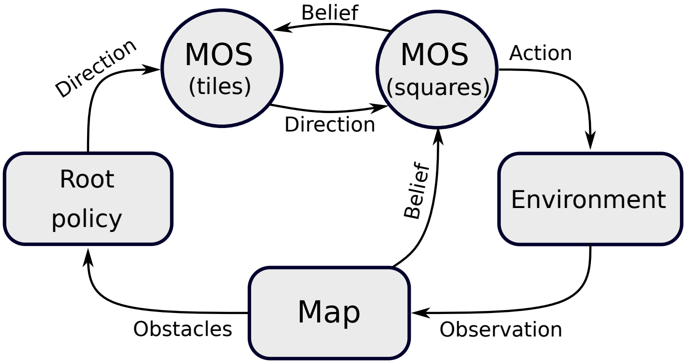


Figure 4.12: The HRL structure implementing the VST-trained policy of table T on two levels of abstraction.

is always initialized optimistically, i.e., with all transition cells as empty, and is updated online with any new information obtained from applying T to the microgrid. Conversely, the belief for the square navigation depends on the map of the agent, and is initialized pessimistically by treating unobserved squares as occupied.

In theory, this abstracted, cascaded structure can be increased to progressively higher levels: *sectors* of 2-by-2 tiles, as well as transitions between sectors, could be abstracted as cells of a macrogrid. The policy of T, applied to these cells, would indicate transitions between sectors. The corresponding action would then initialize multiple "move between tiles" TEA, which in turn would multiple "move between squares" TEA, which finally would be composed by primitives.

However, regardless of the level of abstraction, a root policy must always be present to select a direction to goal for the first subordinate, e.g., a random policy. Here, the hierarchical structure of control (Figure 4.12) is limited to a root policy with two underlying levels of tile navigation (where tiles and connection between tiles are abstracted into cells) and of square navigation (where the only abstraction concerns the trainee agent). Rather than a random policy, the proposed root policy indicates the direction of motion according to a map-based strategy, which is described in the following section.

Exploration strategy

In addition to the local navigation policy given by T, a root policy must be defined or learned in order to efficiently find the goal. This section illustrates how the current map of the environment is used to select *target sectors* for the agent in order to explore.

The strategy adopted for exploration is to individuate *frontiers* [118]. Frontiers are those portions of the border of the current map that are not occupied by obstacles, and that therefore do not prevent movement. The agent must cross a frontier in order to move into unobserved areas of the map, explore the environment further, expand its map, and eventually find the goal. The frontier approach is combined with RL in order to reduce unprofitable exploration. The procedure is

Algorithm 4 Target sector selection

```
Initialize map, $\sigma \in \Sigma$
$V \leftarrow 0$
while goal not found do
    $\text{unexplored} \leftarrow \{\sigma : < 50\%\ observed\ squares\}$
    $\texttt{elig} \leftarrow \{\sigma : \ adjacent\ to\ \text{visited}\} \cap \{\neg\text{visited} \cup \text{unexplored}\} \cap \{\sigma : \ \exists\, frontier\}$
    $\texttt{trgt} \leftarrow \text{argmax}_{\text{sect} \in \texttt{elig}} V(\text{current}, \text{sect})$
    $\texttt{counter} \leftarrow 2 \cdot \text{distance}(\text{current}, \texttt{trgt})$
    while trgt is not reached $\wedge\ \texttt{counter} \neq 0$ do
        $navigate\ towards\ \texttt{trgt}$
        $\texttt{counter} \leftarrow \texttt{counter} - 1$
        $update\ \texttt{map}$
    if trgt is reached then
        $r \leftarrow -0.1$
        $update\ V(\Sigma, \texttt{trgt})$
        otherwise
        $r \leftarrow -1$
        $update\ V(\sigma_{tr}, \texttt{trgt})$
    $update\ \Sigma$
```

summarized in Algorithm 4.

At the start of exploration, the agent has an initial map. The map is then partitioned into groups of 6-by-6 squares, the sectors $\sigma \in \Sigma$. If needed, additional unobserved squares are added to the boundaries of the map until all sectors are exactly 6 squares of side. Initialize the value function $V : \Sigma \times \Sigma \rightarrow \mathbb{R}$ as zeros. All sectors that:

- are adjacent to a *visited* sector, i.e., a sector which contains at least a visited square, and

- are not visited, or contain more than 50% unobserved squares, and

- are separated by a *frontier* from at least one of their neighbors

are *eligible* target sectors. Frontiers are defined as follows: if the agent can reach the candidate sector starting from any side of one of the adjacent sectors, then a frontier exists between the two sectors. Unobserved squares are assumed empty for the sake of determining if a frontier exists. Figure 4.13 shows an example: neighbor sector $\mathbf{A}$ does not have a frontier with the candidate, since the common side cannot be reached due to occupied squares; on the contrary, neighbor $\mathbf{B}$ has a frontier to the candidate.

The actual target among the eligible sectors is then selected according to value function V:

$$\texttt{trgt} = \text{argmax}_{\texttt{elig}} V(\texttt{curr}, \texttt{elig}). \tag{4.10}$$

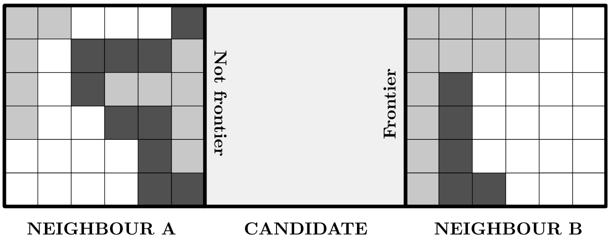


Figure 4.13: An example of frontier evaluation. Black squares are those where the agent has observed an obstacle, while the squares that the agent has observed as empty are in white. Gray squares are then unobserved squares.

If more than one sector has the same value, the nearest one is selected. The use of a value function to select actual targets is motivated by the fact that the agent might not be able to reach the target sector in a reasonable amount of time, e.g., because the unobserved squares turn out to be occupied, or because the agent needs to circumvent a wall in order to reach the target. Therefore, the agent is given a `counter` as a time limit to reach the assigned target. The `counter` starts equal to twice the distance, measured in sectors, between the agent and the target. Each time that the agent is assigned a direction of motion (as in Figure 4.11), the `counter` is reduced by one. If the agent moves to the target before the `counter` reaches zero, it is assigned a reward $r = -0.1$. Then $\forall \sigma \in \Sigma$, V is updated accordingly as

$$V(\sigma, \texttt{trgt}) = V(\sigma, \texttt{trgt}) + 0.2 \cdot (r - V(\sigma, \texttt{trgt})); \tag{4.11}$$

otherwise, $r = -1$ and, given σ_{tr} the sectors visited during the trajectory to `trgt`, it is

$$V(\sigma_{tr}, \texttt{trgt}) = V(\sigma_{tr}, \texttt{trgt}) + 0.2 \cdot (r - V(\sigma_{tr}, \texttt{trgt})). \tag{4.12}$$

Since the reward is always negative, initializing V as zeros is optimistic. Therefore, when a target sector is reached, it is unlikely to be selected again due to Eq. (4.11) regardless of the current sector. Conversely, Eq. (4.12) penalizes the choice of target only for those sectors for which a path is not found. Since the map is continuously updated during navigation, the value function is expanded after each update to include any new sector discovered.

4.4.3. The Parr-Russel maze task with VST-trained HRL agent

The HRL agent, with VST-learned MOS policies, is tested in a specific maze environment (Figure 4.14), originally devised by Parr and Russell [8] as a case study for the HAM approach. The maze is a grid of 85 squares of side, of which approximately 3600 are visitable, while the rest are either occupied or not accessible. The maze contains several identical obstacles that create multiple bottlenecks and cause the agent to waste actions if it enters within the "u" shape of an obstacle. The agent is initialized in the top-left corner of the map at a random cardinal orientation. The task is to find the goal states, which are located at the opposite side of the maze, in the least amount of time.

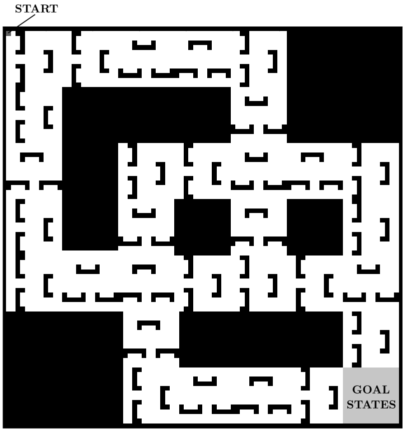


Figure 4.14: The maze devised by Parr and Russell for HAM application (adapted from [8]).

The initial map is entirely composed of unobserved squares, except for the initial sensor readings. Therefore, the agent does not have any prior information on the maze or on where the goal states are, and must explore the maze as fast and efficiently as possible. In order to navigate, the table T is obtained by letting the trainee agent learn in the minigrid environment for 120000 episodes. The hierarchical control described in Section 4.4.2 is then implemented. For the first 100 time-steps, the agent adopts the following stochastic root policy: with probability 0.79, the agent advances in the direction of its current orientation; with probability 0.13 the agent turns counterclockwise; the rest of the times it turns clockwise. This policy is the result of a semi-empirical selection among similar candidate policies, evaluated in a simulated goal-finding task in multiple randomly cluttered environments. Albeit suboptimal, it provides an incentive to exploration compared to a canonical random policy.

After the 100^{th} time-step, the root policy is modified to take into account the presence of the updated map. With probability 0.2, the agent follows the earlier semi-empirical policy; otherwise, a target sector is identified according to Eq. (4.10). When a target sector is assigned, the agent turns to face the cardinal direction that will bring it closest to the target; e.g., if the target is located at North-North East, the agent would turn to face North. It will then advance in that direction until either its orientation is no longer the closest to target (in which case it will adjust its orientation and continue advancing), until it completes the path (updating value function V with a success, and returning to the root policy), or until it fails to reach the target before the `counter` runs out (in which case it updates V with a failure, and again returns to the root policy for a new target). The agent continues exploring until it comes in sensor range of any goal state, which terminates the episode.

4.4.4. Results and conclusions for the maze task

This section discusses the results of the maze task. First of all, during exploration the agent never collides with obstacles or walls. This descends directly from the implementation of the trainee policy, given by $\mathtt{T}$, and applied to the minigrid. During training, actions that lead to collisions are discarded and not included in $\mathtt{T}$, and therefore are not attempted in the maze. It is worth noticing that, while the higher level policies of the root and of the tile MOS are "optimistic", it is sufficient to constrain the lowest level policy, i.e., the "pessimistic" MOS navigating within squares, to obtain a safe hierarchical policy.

Nonetheless, the efficacy of this method is also a result of the absence of discrepancies between the trainee dynamics and the actual dynamics of the robot. Inevitable model discrepancies, as well as sensor noise and odometry errors, must be taken into account when interpreting these results for online exploration. However, a few considerations can be done. First, if the errors in odometry and readings can be estimated, the map can still be used to estimate which, among all obstacle beliefs $\mathcal{O}$, most likely corresponds to the true position of the robot and to the actual obstacle disposition. Second, if a discrepancy between the trainee model transition and the environment is observed, the agent can modify its $\mathtt{T}$ table accordingly, and possibly return the discrepancy to the trainee agent for corrective learning. Finally, even assuming the presence of discrepancies, following the trainee policy is in all likelihood safer than blind search, since the trainee avoids those actions that are certainly unsafe in both the virtual environment and the real one (e.g., moving into a wall).

Other than increasing the safety of exploration, the use of the map-based policy allows the agent to explore efficiently. Figure 4.15 shows the amount of steps to goal and the number of observed squares for 30 sample episodes of the algorithm. Each curve represents a different episode and terminates with a square. The number of primitive actions to goal has a mean value of 1917 (indicated by a vertical line), but it varies from a few hundreds to more than five thousands. This is coherent with the experimental setup, since the agent does not know the location of the goal, but only tries to maximize its exploration. Therefore, when the agent explores in the goal direction by chance, the goal states can be found in as little as 620 timesteps. Conversely, if the agent explores away from the goal, completion time can increase by several times.

To provide a reference, the algorithm is compared with two different hierarchical approaches to the same problem: the original HAMs of Parr and Russel [8], and the results of Zhou et al. [119]. Parr and Russel define a-priori a hierarchy of machines, each representing a constraint on the space of possible policies, connected through `call`, `stop` and `choice` states. The agent considered by Parr and Russel has a different set of sensors: four short range "sonars" which detect obstacles in the adjacent square, and a long range, high-directed sensor that can spot obstacles far away. Another difference is that Parr and Russel's agent does not perform mapping. Under these conditions, the HAMQ-learning achieves an ideal performance after 270000 iterations.

In the work of Zhou et al., a flat Q-learning algorithm is compared to a hierar-

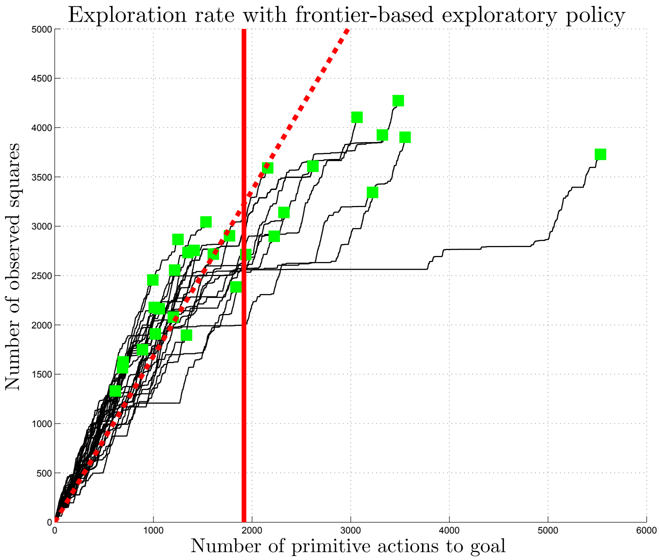


Figure 4.15: The amount of exploration performed by the agent in the Parr-Russel maze with the frontier approach. Each line represents one of 30 different episodes, terminating with a square. The vertical line represents the mean number of primitive actions to goal. The dashed line indicates the mean exploration rate.

chical Q-Learning algorithm, using the same agent described in Section 4.4.2. The higher level of hierarchy of this algorithm uses sensor readings to estimate the current position of the agent with respect to an internal map, which is updated during and after each episode. This constitutes a *belief macro state*, upon which a direction command (e.g., `North`) is sent to the lower level controller, which navigates avoiding obstacles. The initial performance of Q-learning and Hierarchical Q-learning is approximately the same, with primitive actions in the order of 10^5 to find the goal, which reduces drastically to around 5000 primitives for the hierarchical algorithm after 30 episodes.

A direct comparison between the proposed algorithm and the two discussed above should be avoided, due to the intrinsic differences between the three approaches in terms of goals and learning process. Nonetheless, it should be noted that the proposed algorithm is able to achieve a comparable level of performance during its first episode, without prior information on the specific environment, but including off-line training. This proves the effectiveness of implementing VST and state abstraction to obtain safety as well as an improved initial performance with respect to canonical exploratory policies.

In the first 1000 time-steps of Figure 4.15, the exploration rate is very consistent, with a mean of approximately two new observed squares per time-step. In order to appreciate the result, consider that the maximum number of new squares per time-step allowed by the sensor reading is of seven when advancing, and of three

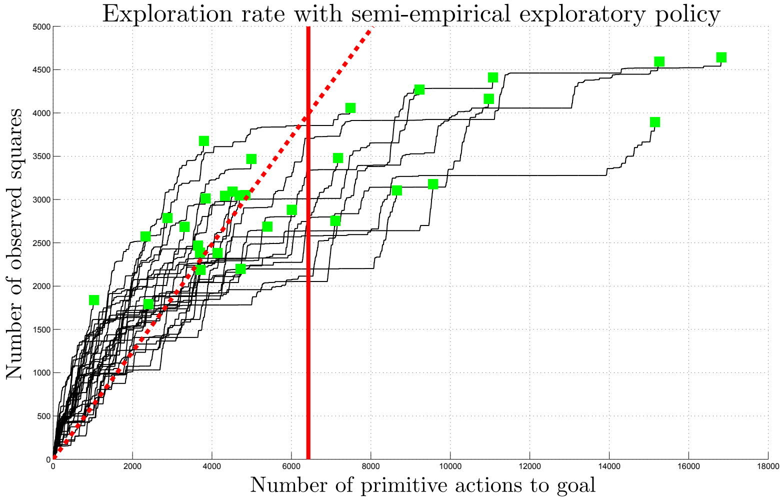
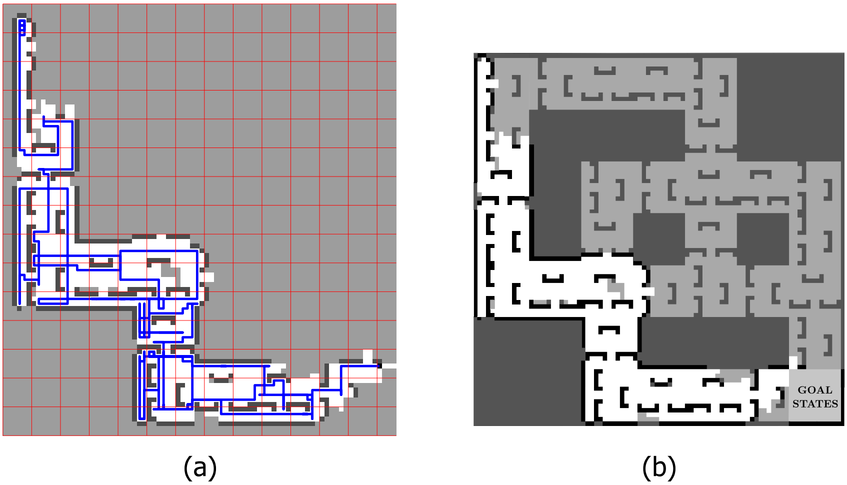


Figure 4.16: The amount of exploration performed by the agent in the Parr-Russel maze with the semi-empirical policy.

when turning, in the event of completely unobstructed, unobserved surroundings. Afterwards, a reduction in exploration can be observed, for which there are two explanations. First, the agent often needs to relocate to another position of the map in order to continue the exploration, navigating through already explored areas of the maze as it does so. Second, the agent often attempts to explore the unobserved states on the other side of walls, treating them as obstacles. These attempts are eventually abandoned by implementing the value function; however they cause the agent to spend actions that do not contribute to the exploration. These explain the peculiar shape of the curves, in which steep rates of exploration are separated by period with little or no new observations. Even when accounting for these effects, the mean rate of exploration is 1.68 new squares per time-step (indicated by the dashed line).

In order to evaluate the exploratory strategy adopted by the agent, Figure 4.16 shows the exploration rate when, instead of adopting the frontier approach, the initial stochastic, semi-empirical root policy is adopted for the entire duration of the task. Once again 30 different episodes are presented. It can be seen that the mean amount of steps to completion is significantly higher, approximately 6400 steps, and that the mean amount of new squares per time-step is also lower, equal to 0.62.

Finally, Figure 4.17a shows the final map and the trajectory of the agent after a typical episode. The agent starts with 100 primitive actions suggested by the semi-empirical stochastic root policy. As a result to this, the first "room" between the starting position and the first bottleneck is only partially explored. Conversely, all remaining areas through which the agent successively navigates are almost completely explored, see Figure 4.17b. The agent's trajectory covers most of the open areas, only occasionally repeating its steps. As an exception, the agent spends



(a) (b)

Figure 4.17: The final map (a) of the agent after one sample episode. The agent starts at an extremity of the map, in the top-left corner, and follows the trajectory until it finds the goals. If the map is compared to the entire maze (b) it can be seen that the agent explores efficiently its surroundings during navigation.

a few timesteps in the proximity of the wall at the bottom-left of the maze. As mentioned, this is due to the agent trying to access the area behind the wall itself, which causes a small delay in exploration. Thus, the proposed algorithm, designed to explore safely, also prevents the inefficiencies that RL and HRL tend to exhibit in the initial phase of learning.

It is worth mentioning that the previous results concern independent episodes, so that during each run, the agent creates a new map and a completely new value function V. However, if the agent does have a previously generated map, it can use this information to reach the goal in a lower amount of iterations. A value function $V_g : \Sigma \to \mathbb{R}$ is learned off-line based on the map, as available at the end of the exploration. A flat RL agent is positioned at the start of an episode in the sector σ_{st} of the map containing the initial position of the robot. The flat agent is then allowed to move directly from one sector to any of the four adjacent ones, as if they were empty adjacent squares, but is not allowed to move into an unexplored sector of the map. Apart from this limitation, the agent follows a random policy, receiving a reward of -1 per move, until it reaches the goal sector, i.e., the one containing the goal, upon which a reward of 100 is issued, and the episode terminates. The value of sectors is updated as usual with the exception of the goal sector, whose value is set as 100.

After a given number of episodes, the value V_g is finalized and can be exploited. Different implementations can be devised. One logical choice consists of using the VST policy once more, so that each cell of the macrogrid corresponds to a sector. Among the sectors corresponding to possible goal cells (Figure 4.18), the one with the highest value is selected as an intermediate sector to the goal. A macrogrid

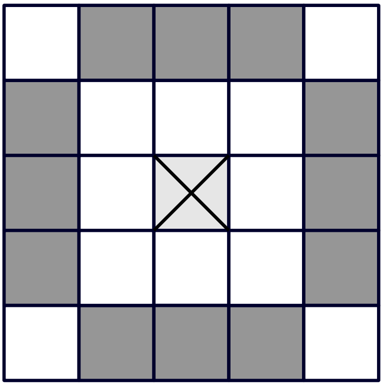


Figure 4.18: From the currently occupied sector (marked with a cross), the agent selects the intermediate sector to goal among those corresponding to the goal cells of a macrogrid (shaded).

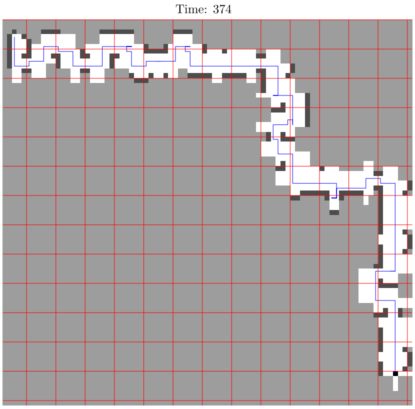


Figure 4.19: A sample follow-up episode with the algorithm. Using the previously gained map and the location of the goal, this can be reached with less than 400 primitive actions.

is positioned as to include the current sector and the intermediate one, as a goal cell. The policy of $\mathtt{T}$ is then used to issue a direction to goal to the MOS, effectively replacing the root policy. Figure 4.19 shows a follow-up episode, with the value V_g derived from a previous map. In this case, less than 400 primitive actions are needed to reach the goal again from the starting position of the robot. Therefore, this HRL implementation not only results in safe and efficient exploration, but can also use previous knowledge to further improve its policy.

In conclusion, the application of VST and abstraction to design an HRL agent with a constrained, feudal MOS policy results in safe and efficient exploration of the unknown but discrete maze environment. Furthermore, this policy has an initial performance that, albeit suboptimal, is not affected by the conventional "blind search" phase, compares satisfactorily with similar HRL implementations, and can reuse previous knowledge of the environment to accelerate the completion of the task.

4.5. VST with non-exhaustive beliefs and uncertain dynamics

The maze navigation of Section 4.4 illustrated how to implement SHRL, state abstraction and VST in a discrete environment. Even though the method could serve as a basis for an actual robotic application, the discrete maze setting is more reminiscent of a classic RL gridworld [71] than of a real-life UAV environment. Section 4.5 presents a more realistic application of VST to a problem with a continuous environment. In particular, a flight task is simulated where an MAV flies in an environment containing obstacles whose shape, disposition and number is unknown. Additionally, the only dynamics available are those of an uncertain bounding model.

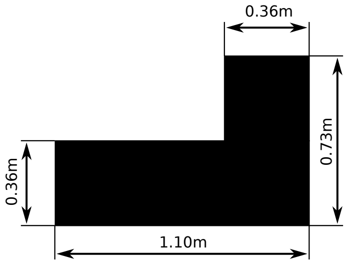


Figure 4.20: Shape of elementary obstacles.

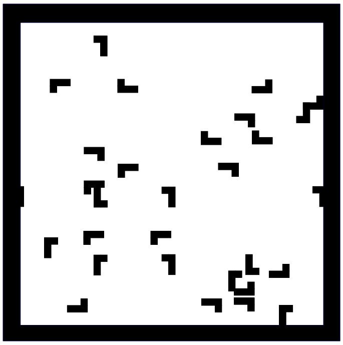


Figure 4.21: Example of a room environment for the task.

This task represents therefore a harder challenge for the application of VST with respect to the case of Section 4.4.

4.5.1. Task and MAV dynamics description

Task description

Similarly to the maze task, the MAV task consists of finding an unknown goal position within an indoor environment containing obstacles. However, the environment of this task differs from the Parr-Russel maze in that there is no predefined and discrete grid of locations. Instead, every episode of the task is performed in a different *room* environment. All rooms are approximately 20m by 20m in size and are delimited by walls, but differ in that they contain different, complex obstacles. These are obtained by sparsely and randomly positioning 30 elementary obstacles with the shape of Figure 4.20. Elementary obstacles are flipped and rotated, so that they overlap into formations as shown in Figure 4.21. Each room also contains one randomly placed goal in an unknown but empty position.

A collision occurs whenever the MAV, which for simplicity is considered a 2D point-mass, occupies the same position of an obstacle or a wall, which in this task stands for a fatal occurrence. As in the maze task, the agent has sensors that can localize nearby obstacles, which are used to prevent collisions; however, due to constraints in terms of acceleration and turning rate of the MAV, this task's environment contains LTF states, which were not present in the maze task. Each episode then lasts until either a collision occurs, or until the agent is within $1\mathrm{m}$ from the goal.

MAV description

The bounding model of the MAV dynamics with state (x_1, x_2, χ, V) is

$$\dot{x}_1 \in \hat{\eta} V \sin(\chi) \; ; \; \dot{x}_2 \in \hat{\eta} V \cos(\chi) \; ; \; \dot{\chi} \in \hat{q} \cdot u_1 \; ;$$
$$\dot{V} \in \begin{cases} \max(0, \hat{a} \cdot u_2) & \text{if } V = \underline{V} \\ \hat{a} \cdot u_2 & \text{if} V \in (\underline{V}\,, \overline{V}) \\ \min(0, \hat{a} \cdot u_2) & \text{if } V = \overline{V}, \end{cases} \tag{4.13}$$

where (x_1, x_2) is the absolute position of the MAV, χ is the heading, and V is the speed, which is comprised between $\underline{V} = 0.1\frac{\mathrm{m}}{\mathrm{s}}$ and $\overline{V} = 0.3\frac{\mathrm{m}}{\mathrm{s}}$. The MAV actuation allows a maximum yaw rate of $30\frac{\circ}{\mathrm{s}}$, and a maximum acceleration or deceleration of $\pm 0.3\frac{\mathrm{m}}{\mathrm{s}^2}$.

The parameters $\hat{\eta} = [0.9\,, 1.1]$, $\hat{q} = [17\frac{\circ}{\mathrm{s}}\,, 30\frac{\circ}{\mathrm{s}}]$, $\hat{a} = [0.1\frac{\mathrm{m}}{\mathrm{s}^2}\,, 0.3\frac{\mathrm{m}}{\mathrm{s}^2}]$ are intervals, so Eq. (4.13) is effectively a distribution of models. Each combination $\eta \in \hat{\eta}$, $q \in \hat{q}$ and $a \in \hat{a}$ is a *realization* of the bounding model.

Sensors onboard individuate obstacles at a distance of $1.5\mathrm{m}$ and at an angle comprised between $-100°$ and $100°$ of the current orientation.

4.5.2. SHRL agent design for the MAV task

As Figure 4.22 shows, the SHRL agent for this task consists of two MOS: a supermanager (*SupM*) and a submanager (*SubM*). *SupM* is tasked with assigning a temporary destination to *SubM*, among the unvisited portion of the room, while *SubM* must bring the MAV closer to the destination, as well as avoid collisions.

The main motivation to separate the two managers is to constrain the overall policy towards safety of exploration. This is done by applying VST to *SubM*, the lowest-level MOS. This constrains the set of discoverable policies, similarly to what presented in Section 4.4 for the maze task agent.

Additionally, this separation is motivated by the difference between the two constituents of the environment: the room and the MAV itself. The objective of *SubM* is to fly the MAV away from obstacles and towards a target heading: the knowledge of the approximate dynamics of the MAV can be used to enforce this. Conversely, *SupM*'s objective is unrelated to the disposition of the obstacles, as it only concerns the position of the goal. Furthermore, while the policy of *SubM* is constrained in order to achieve safety via SHRL, the policy of *SupM* is not, to guarantee that the goal is eventually found.

SupM is a MOS with the objective of finding the position of the goal. The initialization condition of *SupM* is simply $I = \mathcal{S}$ (since *SupM* is the root). The policy π_{SupM} is defined by a value function similar to the one applied in Section 4.4.2 for the exploration strategy. The room is partitioned into a `grid` of $2\mathrm{m}x2\mathrm{m}$ squares. The value function V_{SupM} : `grid` $\rightarrow \mathbb{R}$ is initialized optimistically and used to assign to *SubM* a square of `grid` as the direction to goal. The MOS terminates if either the square indicated by V_{SupM} is reached, or if a `counter` indicates that a set amount of primitives has been executed. After termination, reward is assigned to update V_{SupM}, as detailed in Section 4.5.4.

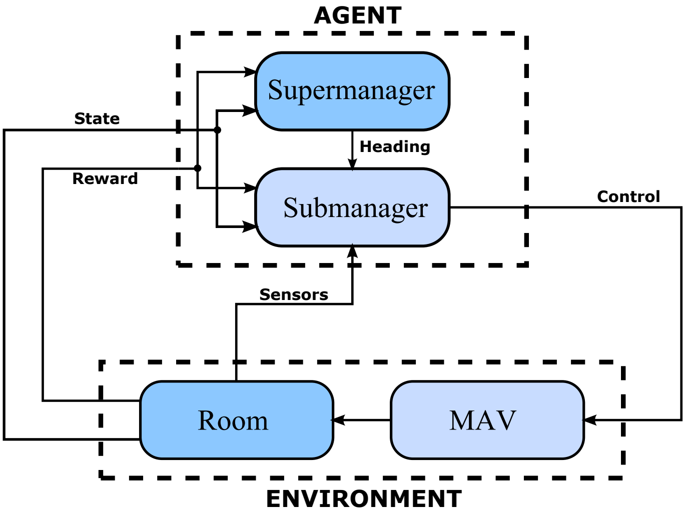


Figure 4.22: The two level managerial architecture of the agent. *SupM* assigns the heading to *SubM*, which in turn control the MAV with the aid of sensor information. The color association indicates the focuses of the two managers: the room (i.e., the goal) for *SupM*, the MAV flight for *SubM*.

The MOS *SubM* is initialized exclusively when called by *SupM*. The policy π_{SubM} has two additional inputs other than the state $\boldsymbol{x}$ (Figure 4.22), i.e., the heading from *SupM* and the "immediate" sensor reading at the time-step k^{in} when the MOS is initiated. These two elements constitute the *belief* of *SubM*. Then, *SubM* terminates upon collision or when the MAV exits the portion of room that at time-step k^{in} was in range of the sensors.

4.5.3. VST for the MAV task

This section illustrates how VST is applied to the MAV task, including the necessary assumptions.

State projection

The MOS policy $\pi_{SubM} = \pi_{\text{V}}$ is learned during VST in order to minimize the risk of collision. As a first step to apply VST, the state is projected according to a function $\mu : \mathcal{S} \to \mathcal{P}^1 \times \mathcal{P}^2$ in the form

$$\begin{aligned}\rho &= \| (x_1 - x_1|_{\text{c}}, x_2 - x_2|_{\text{c}}) \|;\\ \theta &= \arctan\left(\frac{x_2 - x_2|_{\text{c}}}{x_1 - x_1|_{\text{c}}}\right) - \chi|_{\text{c}};\\ \psi &= \chi - \chi|_{\text{c}} - \theta,\end{aligned} \tag{4.14}$$

with $\boldsymbol{p}^1 = (\rho, \theta, \psi, V)$ and $\boldsymbol{p}^2 = (x_1|_{\text{c}}, x_2|_{\text{c}}, \chi|_{\text{c}})$. Terms ρ and θ describe the relative position of the MAV in cylindrical coordinates with respect to the reference

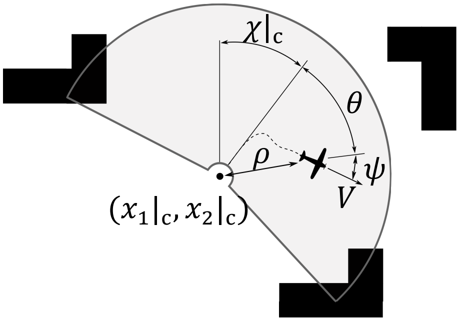


Figure 4.23: The change between the absolute reference system (x_1, x_2, χ) of *SupM* and the relative reference system (θ, ρ, ψ) of *SubM* depends on the absolute position and orientation $(x_1|_\mathrm{c}, x_2|_\mathrm{c}, \chi|_\mathrm{c})$ at the time k^in of subtask assignment. Speed V is the same in the two systems.

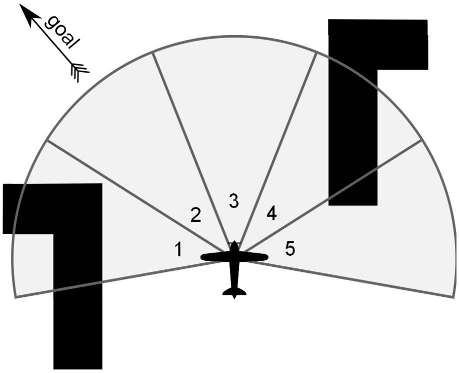


Figure 4.24: The sensory range is split into sectors 1 to 5. In this example, sectors 1, 4 and 5 are blocked as they contain at least one obstacle, and sector 2 is the target sector.

$x_1|_\mathrm{c}, x_2|_\mathrm{c}$; ψ indicates the heading. Figure 4.23 shows this projection. It is important to notice that the projected state $\boldsymbol{p}^1$ is relative to and independent of $\boldsymbol{p}^2$.

Agent belief through sector state abstraction

The projected fatal states are all those tuples $(\rho\,,\,\theta)$ for which a collision occurs, given the obstacle disposition. In order to perform VST, a belief set of these obstacles, as well as of the direction to goal, must be defined. Contrarily to the minigrid environment of Section 4.4.2, it is infeasible to exhaust all possible dispositions of obstacles within the sensor range, due to the continuity of the environment, and due to the variability in shape of the obstacles. It is therefore not possible to obtain an exhaustive belief set as in the maze environment.

However, a belief set $\mathcal{B}$ which approximates the room environment is obtained by abstracting the sensing range into *sectors*, as shown in Figure 4.24. Sectors are numbered from one to five and are $40°$ wide. If the sensors detect an obstacle inside a sector, then the sector is said to be *blocked*, otherwise it is *empty*. Additionally, sectors convey the direction to goal assigned by *SupM*: the sector containing or nearest to the assigned direction of flight is the *target* one. This target sector is also part of the belief. The use of sectors as representatives of fatal states is a rough abstraction of the presence of obstacles. However, reducing the complex sensor range to a sector representation has the benefit of limiting the elements of $\mathcal{B}$ to a manageable amount. The goal of the policy π_V, to be learned with VST, is to avoid blocked sectors and to reach the target sector, if this can be done safely.

Virtual Safety Training

From the uncertain model of Eq. (4.13), a "projected model" $\hat{\mathcal{D}}^1$ is derived:

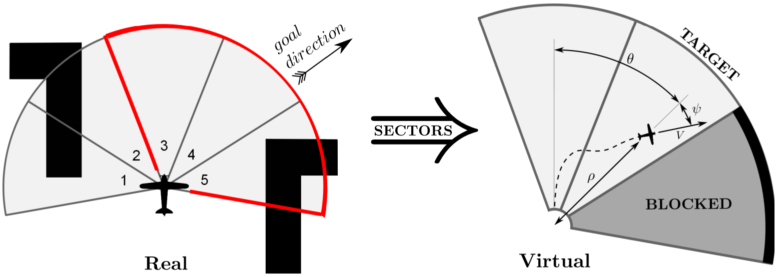


Figure 4.25: The virtual environment (right) is composed by three sectors equivalent to those over which policy π_{SubM} is defined. Sectors that contain an obstacle are blocked, sectors containing the direction of flying are target sectors. Restricting the virtual environment to only three sectors allows to reduce the amount of possible subtasks, of which the figure on the left is only an example, to a only 24 cases.

$$\dot{\theta} \in \hat{\eta}\frac{V}{\rho}\sin\psi \ ; \ \dot{\rho} \in \hat{\eta} V \cos\psi \ ; \ \dot{\psi} = \begin{cases} \max(0, \hat{q} \cdot u_1 - \dot{\theta}) & \text{if } \psi = \underline{\psi} \\ \hat{q} \cdot u_1 - \dot{\theta} & \text{if} \psi \in (\underline{\psi} \, , \overline{\psi}) \\ \min(0, \hat{q} \cdot u_1 - \dot{\theta}) & \text{if } \psi = \overline{\psi} \end{cases} \ ; \tag{4.15}$$

$$\dot{V} = \begin{cases} \max(0, \hat{a} \cdot u_2) & \text{if } V = \underline{V} \\ \hat{a} \cdot u_2 & \text{if} V \in (\underline{V} \, , \overline{V}) \\ \min(0, \hat{a} \cdot u_2) & \text{if } V = \overline{V}. \end{cases}$$

Notice that an additional state constraint $\psi \in [\underline{\psi} \, , \overline{\psi}] = [-90° \, , 90°]$ is imposed. This constraint on ψ prevents the MAV from flying in circles indefinitely and forces it to eventually exit the virtual environment.

In order to further reduce the learning space of π_{V}, the virtual environment is limited to only three sectors, corresponding to 3, 4 and 5 of Figure 4.24. The motivation for this is that the model of Eq. (4.15), as well as the placement of the sectors, are symmetrical with respect to the axis $\theta = 0$. Therefore, a policy π_{V} for the *right side* of the sensor range (sectors 3, 4 and 5) can also be applied to the *left side* (sectors 1, 2 and 3) by inverting the yaw rate control u_1. However, a consequence of this is that the agent must be provided a criterion to choose which of the two sides to fly in. With this choice in effect, the virtual environment is then delimited by

$$\theta \in [\underline{\theta} \, , \overline{\theta}] = [-20° \, , 100°], \rho \in [\underline{\rho} \, , \overline{\rho}] = [0.1 \, , 1.5].$$

As a second simplification, the policy actions are reduced to $u_1, u_2 \in \{-1; 0; 1\}$.

The belief set is then a combination $(\mathtt{obs}_1, \mathtt{obs}_2, \mathtt{obs}_3, \mathtt{goal}) \in \mathcal{B}$ where $\mathtt{obs}_i$ is true if the corresponding sector is blocked and false otherwise, and $\mathtt{goal} \in \{3, 4, 5\}$ identifies the target sector. An example of the room environment and relative belief is shown in Figure 4.25. The lower number of sectors allows to reduce the number of elements in $\mathcal{B}$ sectors from $5 \cdot 2^5 = 160$ to $3 \cdot 2^3 = 24$. Summarizing, the VST policy is defined as

$$\pi_{\mathrm{V}} : \mathcal{P}^1 \times \mathcal{B} \to \mathcal{A} = \{-1, 0, 1\} \times \{-1, 0, 1\}. \tag{4.16}$$

Algorithm 5 shows the VST procedure for π_{SubM}. Three look-up tables Q_g, Q_o, and $\mathtt{Vis} : \mathcal{P}_d \times \mathcal{A} \to \mathbb{R}$ are initialized, with all entries equal to zero. Q_g and Q_o are two state-action value functions indicating respectively the value towards reaching the target sector and the value towards avoiding obstacles. $\mathcal{P}_d$ is a discrete state space obtained by partitioning the components of $\mathcal{P}^1$ into intervals: $[\underline{\theta}, \overline{\theta}]$ and $[\underline{\rho}, \overline{\rho}]$ are partitioned into nine intervals, $[\underline{\psi}, \overline{\psi}]$ in seven interval, and $[\underline{V}, \overline{V}]$ into four, for a total of 54432 aggregated states. The table Vis is used to store the amount of times that an aggregated state-action pair is visited during learning.

Since the model of Eq. (4.15) is not "crisp", it cannot be directly utilized to simulate trajectories; instead, a realization η, q, a is randomly assigned from the distributions of Eq. (4.15) at the start of each episode. Thus, the VST will yield the policy π_{V} that is optimal on average. Also, due to the episodic nature of learning, a Monte Carlo method [4] is applied.

4

At the start of each "virtual" episode, the state of the environment and the belief of the agent are drawn randomly from the candidate set

$$[-10^\circ\,,\,10^\circ] \times [\underline{\rho}\,,\,\underline{\rho} + 0.1\mathrm{m}] \times [-10^\circ\,,\,10^\circ] \times [\underline{V}\,,\,\overline{V}] \times \mathcal{B}, \tag{4.17}$$

and the episode stops when the MAV leaves the environment, i.e., $\theta \notin [\underline{\theta}\,,\,\overline{\theta}]$ or $\rho \notin [\underline{\rho}\,,\,\overline{\rho}]$. Note that due to the constraints on ψ, all simulations will be of finite duration.

During VST, an ϵ-greedy exploratory policy is applied, with $\epsilon = 0.8$; for this purpose, greediness is defined with respect to the state-action value function Q_g. The first time an aggregated state-action pair is visited during an episode, it is appended to a temporary list. Reward is then assigned to all the elements of list depending on the termination condition. If the MAV exits through the arc ($\rho > \overline{\rho}$) of a blocked sector, tuples are assigned a penalty $r_o = -1$ counting towards the value function Q_o and a zero reward $r_g = 0$ valid for Q_g. The penalty $r_o = -1$ is assigned even if the blocked sector is also the target sector. Instead, if the MAV exits through the target sector, and this is not blocked, rewards $r_g = 10^3$ and $r_o = 0$ are assigned. If the sector is neither blocked nor the target, zero rewards are assigned. If $\theta \notin [\underline{\theta}; \overline{\theta}]$, $r_o = 0$ and a penalty $r_g = -n^{\mathrm{st}}$ are assigned, where n^{st} is the number of times-steps since the start of the episode. At the end of each simulation, the value of all tuples in list is averaged based on the current total visits:

$$Q_*(\mathtt{list}) = \frac{Q_* \cdot (\mathtt{Vis}(\mathtt{list}) - 1) + r_*}{\mathtt{Vis}(\mathtt{list})}. \tag{4.18}$$

where Q_* stands for Q_o or Q_g with corresponding reward r_o or r_g. The temporary list is then depleted, and a new episode is performed as described above.

VST must result in a policy that is safe. A state-action tuple that has led to a collision should be avoided, if possible. This is reflected in the way that Q_g and Q_o are weighted to form a single state-action value function

$$Q = Q_g + \omega \cdot Q_o, \tag{4.19}$$

where $\omega \gg \max(Q_g)$. Then, π_{SubM} is chosen as the greedy policy with respect to Q. This choice optimizes performance for state-actions that are provably safe ($Q_o(\boldsymbol{p}^1) = 0$) and safety for state-actions that are potentially unsafe ($Q_o(\boldsymbol{p}^1) \neq 0$) . In practical terms, adopting $\omega = 10^5$ suffices for this application.

Algorithm 5 VST with Monte Carlo simulations

```
**Initialize**
$Q_g, Q_o \leftarrow$ empty
$\mathtt{Vis} \leftarrow$ empty
**while** training **do**
    $\boldsymbol{p}$, belief, $\{q, a, \eta\}$ randomly initialized
    $\mathtt{list} \leftarrow \{\ \}$
    **while** $\boldsymbol{p}^1 \in \mathcal{P}^1$ **do**
        $u_1, u_2 \leftarrow \epsilon$-greedy$(Q_g)$
        $\mathtt{Vis}(\boldsymbol{p}^1, u_1, u_2) \leftarrow \mathtt{Vis}(\boldsymbol{p}^1, u_1, u_2) + 1$
        $\mathtt{list} \leftarrow \mathtt{list} \cap \{\boldsymbol{p}^1, u_1, u_2\}$
        $\boldsymbol{p}^1 \leftarrow \boldsymbol{p}^1 + \Delta\boldsymbol{p}^1(\boldsymbol{p}^1, u_1, u_2)$
    $r_* \leftarrow$ reward
    $Q_*(\mathtt{list}) = \frac{Q_* \cdot (\mathtt{Vis}(\mathtt{list}) - 1) + r_*}{\mathtt{Vis}(\mathtt{list})}$
```

Preliminary results of VST policy implementation

A total of 60000 virtual learning trials are performed requiring around 40 minutes of computation[2]. The virtual policy π_{V} is then utilized as the submanager policy π_{SubM}. The value function V_{SupM} is initialized as zeros, so that its policy is initially random.

The combined SHRL agent is tested for several episodes with multiple instances of the room environment. At the start of each episode, the MAV is placed inside the room in a random empty position, with random speed $V \in [\underline{V}, \overline{V}]$. A goal square and a realization η, q, a are also randomly assigned. If the target heading assigned by *SupM* is within sectors 3, 4 or 5, then *SubM* flies the MAV in the right side of the sensor range; otherwise, in the left side. When *SubM* terminates, it is initialized again with an updated belief, unless *SupM* also terminates. In this case, first *SupM* assigns a new heading, and then *SubM* receives the new belief.

Results of the test are as follows. First, the controller consistently avoids isolated obstacles: as soon as one is detected, the MAV manages to modify its route in order to avoid a collision. However, when navigating inside a more cluttered area, it can happen that *SubM* terminates in close proximity a previously undetected obstacle. The MAV is then likely to collide during the following execution of *SubM*. This is not

[2] on an Intel Core i5-3360M CPU, 2.80GHz

surprising since π_{V} does not discriminate between "near" and "far" obstacles, but acts only according to its sector belief.

A second observation is that selecting which side to navigate based exclusively on the location of the target is detrimental to both safety and performance. If between the MAV and the target square there are numerous obstacles, it is not only unsafe to navigate in that direction, but also inefficient, since π_{SubM} repeatedly performs evasive maneuvers. Deviating from the target flight direction would then be beneficial, and would allow the agent to discover an easier path to the target square.

As a final remark, the policy π_{SubM} is found to almost exclusively select $u_2 \in \{-1, 0\}$, i.e., to either brake or maintain flight speed V. This is also to be expected as, with the dynamics of Eq. (4.15), flying at a lower speed is generally safer as it allows the MAV more time-steps to turn away from blocked sectors.

Two modifications are adopted based on the previous remarks. First, the termination condition of *SubM* is modified to include changes in belief. At each time-step k, a new sensory information is acquired, as the MAV moves, and a new obstacle belief $\mathtt{b}_{obs}(k) = \{\mathtt{obs}_1(k), \mathtt{obs}_2(k), \mathtt{obs}_3(k)\}$ can be obtained. If for any k, $\mathtt{b}_{obs}(k) \neq \mathtt{b}_{obs}(k-1)$, then *SubM* is terminated. As can be seen from Figure 4.26, this allows the policy to react to newly observed obstacles; additionally, it reduces the amount of unnecessary deviations resulting from obstacles that are only marginally in range and can be avoided in a few time-steps.

As a second modification, the side of flight selected is changed to the one which contains the least amount of blocked sectors, regardless of whether it contains the target sector or not. In the event that the number of blocked sectors is the same in the two sides, the one which contains the target sector is selected. It can be argued that this is in line with the "safety first" concept of this managerial architecture. Figure 4.26 presents an example of policy implementation. Whenever the selected side does not contain the target sector, the middle sector of that side (i.e., sectors 2 or 4 of Figure 4.24), is appointed as a temporary target sector for the application of the policy.

The reward during training is altered to reflect this change in strategy. If $\theta > \overline{\theta}$, a penalty $r_g = -n^{\mathrm{st}}$ is assigned as before; however, if the termination condition is $\theta < \underline{\theta}$, the reward assigned is $r_o = -1$. This alteration is introduced to penalize the MAV if it flies into the discarded side of flight, which is likely to have more blocked sectors, and thus to contain more obstacles.

An additional termination conditions is added as $\rho \geq (\underline{\rho} + \overline{\rho})/2$; as a result, *SubM* is called and reinitialized more often than in the test implementation. This is done to reduce the amount of deviation of the MAV heading from the direction to goal, as indicated by *SupM*, whenever the sectors of the side of flight does not contain the target sector. Policy π_{SubM} is learned a second time in VST, with this new reward function. Section 4.5.4 shows the result of this second implementation of the SHRL agent in the MAV task.

Figure 4.27 shows an example of policy implementation with the previous modifications. The asterisk at the bottom left of the figures represents the temporary target square selected by *SupM*. In Figure 4.27a, the MAV (whose position is rep-

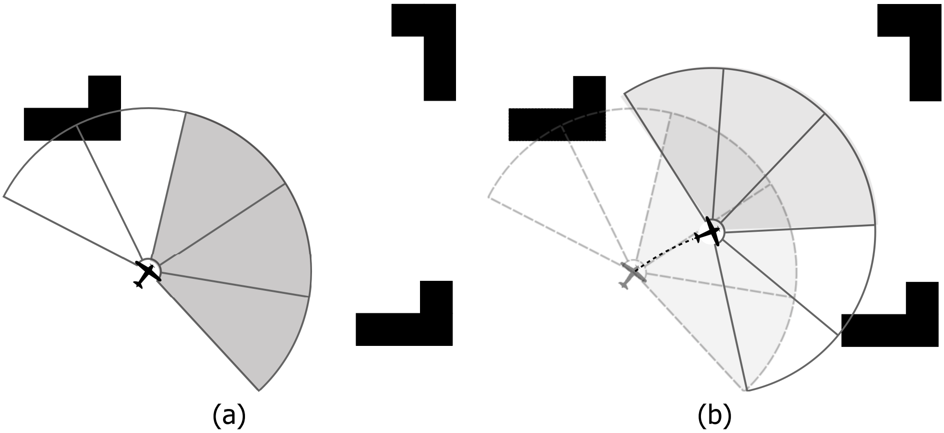


Figure 4.26: An example of policy implementation and termination condition. In (a), *SubM* flies the MAV in the right side, as no obstacle is perceived: $\texttt{obs}_1 = \texttt{obs}_2 =, \texttt{obs}_3 = \texttt{false}$. In (b), however, an obstacle is detected, so that $\texttt{obs}_2 = \texttt{true}$. *SubM* terminates, and is initialized again by *SupM*. Now the MOS selects the left side, as it is free of obstacles.

resented by dots) has initially no obstacle in sensor range (delimited by the two symmetrical lines at the side of the dot). The direction to goal is in the right side of the range, but since an obstacle is detected, *SubM* flies the MAV in the left side until the initial position of Figure 4.27b is reached, upon which *SubM* terminates. Then, it selects the right side sectors as they are the nearest ones to the goal, and successfully flies between the two obstacles.

4.5.4. Results of the MAV task

This section presents the behavior of the MAV SHRL agent during 100 episodes. At the start of the first episode, the MOS *SubM* is initialized with $\pi_{SubM} = \pi_{\mathrm{V}}$, and with the initialization and termination conditions detailed in the previous section. Conversely, the value function V_{SupM} of *SupM* is initialized as zeros and updated online during the execution of the task.

The results shown are relative to the room environment of Figure 4.28, and an MAV model realization with $\eta = 1.09$, $q = 29.04$, $a = 0.203$, but the results generalize to all observed combinations of rooms and model realizations. Position, orientation and speed of the MAV are initialized randomly at the start of each episode, which terminates when the MAV is within $1\mathrm{m}$ of the goal, or if a collision occurs. The target is positioned greedily according to *SupM*'s value function.

The reward assigned to update V_{SupM} is as follows. If the primitive actions' `counter`, which starts at 500, runs out before the target square is reached, $r_{\mathrm{cro}} = -2$ is given, otherwise it is $r_{\mathrm{trg}} = -1$. Also, when an episode ends, it is $r_{\mathrm{end}} = 10$ if the goal is found or $r_{\mathrm{end}} = -5$ if a collision occurs. The following learning parameter are arbitrarily adopted: rate $\alpha = 0.4$, discount $\gamma = 0.8$, and eligibility decay $\lambda = 0.75$.

Figure 4.28 presents the room environment, with the star indicating the goal,

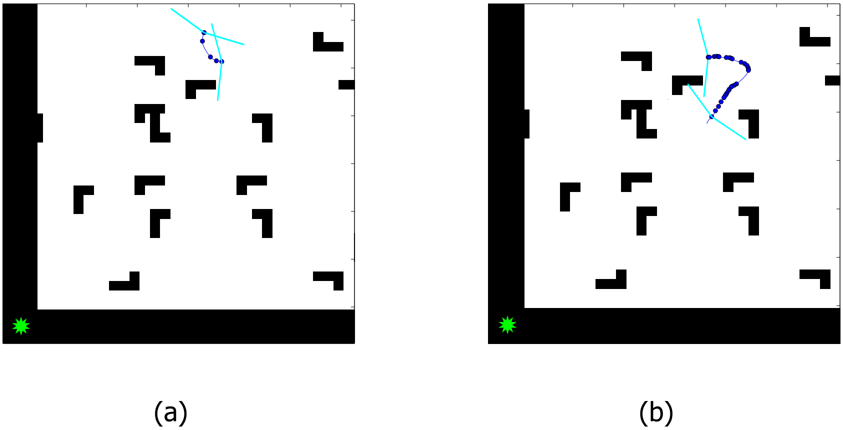


Figure 4.27: An example of obstacle avoidance inside the room environment. The dots represent MAV positions at which sector selection is performed. *SupM* assigns the target square indicated by the asterisk. In (a), *SubM* deviates from the heading assigned from *SupM* as it contains obstacles; after these have been avoided (b), *SubM* terminates and is immediately re-executed by *SupM* with the same goal. This time, *SubM* selects the right side of the range, thus flying nearer to the target.

and the 100 learning trajectories generated during the episodes. It can be seen that π_{SubM} promotes safety, privileging portions of the room that are free of obstacles. In particular, although several paths are available to reach the goal from any point, most of them are rarely if ever taken since they are too narrow for the policy. This happens regardless of the *SupM* positioning of the target square, exemplifying the inherent robustness of the SHRL approach.

The agent manages to reach the goal in 52 out of the 100 episodes. In terms of number of collisions per time-step, Figure 4.29 shows the ratio of collisions to actions. It can be seen that collisions are in the order of one every 2000 steps, equivalent to 400s. It can be seen that the collision rate does not decrease with the episodes, as the *SubM* MOS is already defined at the start of the task. The VST policy π_V is therefore unable to prevent all collisions. However, trajectories are shown to avoid densely cluttered areas of the room, as well as individual obstacles, which can be considered a cautious behavior.

SupM is able to learn the position of the goal. This can be seen not only in the trajectories of Figure 4.28, which tend to coil into a "ring" around the three obstacles at the right of the goal, but also in the value function V_{SupM}, that shows higher value for the entries corresponding to the goal and to the nearby "ring" location. Figure 4.30 shows four additional trajectories, obtained after the 100^{th} learning episode. Each figure shows two trajectories, with the position of the MAV indicated with triangles in one trajectory, and with dots in the other. From four different initial states, and with four different model realizations, the MAV is directed towards the learned goal position by *SupM*. Multiple deviations are necessary to avoid the obstacles until a breach in the obstacles is found. Since the goal is in

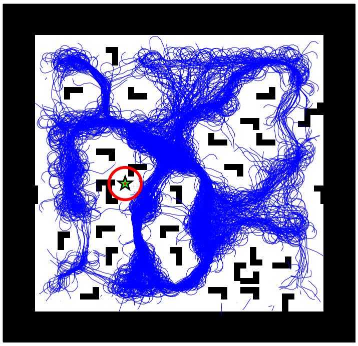


Figure 4.28: The trajectories of the 100 training episodes of the SHRL algorithm. The circle around the goal (represented by a star) indicates the distance that the MAV must reach to complete the task.

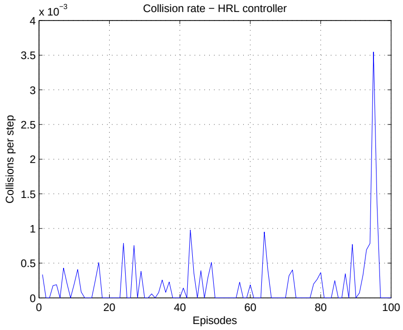


Figure 4.29: The amount of collisions per steps for the SHRL algorithm.

between obstacles, the MAV needs to turn before it can reach its target. It can be seen that the SHRL agent generates consistent, risk-avoiding and goal-finding trajectories, even with the aforementioned differences in initial state and model realizations.

Result validation with flat RL agent

In this section, the results are compared to those of a flat RL agent. With respect to the SHRL agent, the flat one is allowed to learn entirely with the actual room environment and model realization. All episodes are performed with the obstacle disposition shown in Figure 4.28. The same learning parameters of α, γ and λ are employed. Each episode terminates if the goal is reached. In the event of a collision, the MAV is randomly repositioned in the room, and the episode continues. During learning, a reward of 100 is assigned when finding the goal, and of -5 when hitting an obstacle.

The learning space of the action-value function Q_{flat} is the combination of the original, non-projected state $\boldsymbol{x}$ with the addition of the MAV sensor reading. In order to accelerate and simplify learning, the learning space is reduced to a minimum. First, speed V is removed from the learning space on the basis that, as the SHRL agent test demonstrated, the safest action consists in braking towards $\underline{V}$, regardless of the speed.

Second, x_1 and x_2 are partitioned into 20 equally wide intervals, and χ is partitioned into 8 intervals, each $45°$ wide, and aligned with the cardinal directions *North*, *North-East*, and so on. The presence of obstacles in the environment is abstracted through the 5 sectors of Figure 4.24. Sensor readings $\mathtt{obs}_i$ are then included in the learning space of Q_{flat}, for a total of 102400 states.

Third, for the first 1000 episodes, the agent learns a "reduced" action-value function

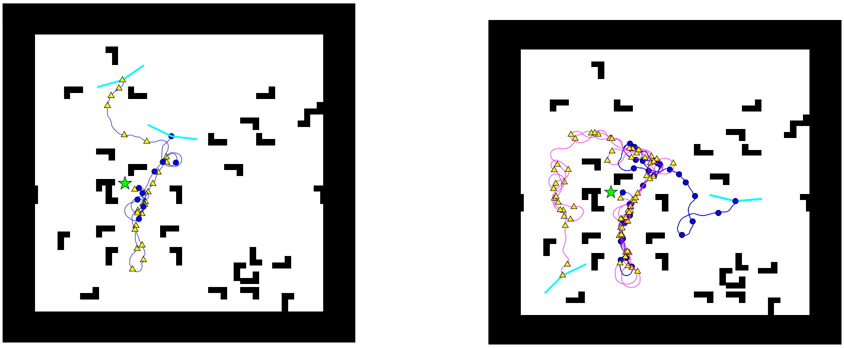


Figure 4.30: Four sample trajectories as generated by the SHRL agent after learning. In all trajectories, the controller avoids obstacles, until he manages to get in between two of the obstacles. The MAV flies then towards the bottom of the room in order to invert direction of flight.

$$Q_{\text{red}}(x_1, x_2, \chi, u_1, u_2),$$

which is initialized as zeros and which does not depend on the sensors readings $\texttt{obs}_i$. This reduces the number of discrete states to 3200, which accelerates learning. This choice is validated by the fact that, since the obstacle disposition is fixed, the sensors readings are in principle only functions of the absolute position and of the heading. During learning, actions are selected ϵ-greedily, with $\epsilon = 0.2$.

After the 1000^{th} episode, the values of Q_{red} are used to initialize the "expanded" action-value function

$$Q_{\text{exp}}(x_1, x_2, \chi, \texttt{obs}_i, u_1, u_2),$$

which also depends on sensor readings. The reason to include $\texttt{obs}_i$ in the learning space is that, while two readings are the same for the same continuous absolute position and heading, this is not true once the state is discretized. A total of 1250 additional learning episodes are performed for Q_{exp}.

As a final measure towards accelerating learning, as proposed by Santamaria and Sutton [120], a third action-value function

$$Q_{\text{obs}}(\texttt{obs}_i, u_1, u_2),$$

is added to Q_{exp}. Q_{obs} is learned in the same way as the previous functions, except that only the collision reward of -5 is assigned for learning. The reason to include this second value function (initially zero) is to learn which actions are dangerous from sensor information, e.g., turning towards obstacles. This knowledge is then generalized over the whole learning space. Therefore, between the 1000^{th} and the 2250^{th} episode, the flat agent learns a combined policy $Q_{\text{flat}} = Q_{\text{exp}} + Q_{\text{obs}}$; an ϵ−greedy policy, $\epsilon = 0.2$, is followed with respect to Q_{flat}.

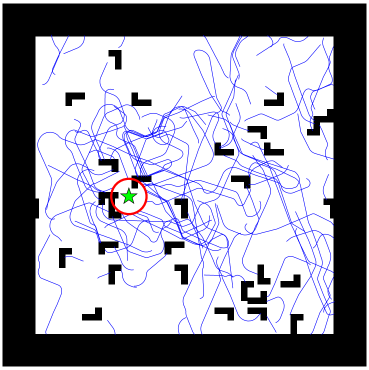


Figure 4.31: The trajectories of 100 episodes after learning of the flat RL algorithm. The circle around the goal (represented by a star) indicates the distance that the MAV must reach to complete the task.

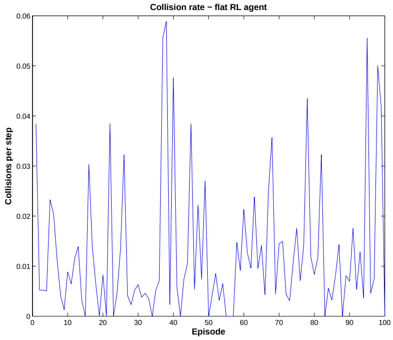


Figure 4.32: The amount of collisions per steps during 100 episodes for the flat RL algorithm after learning.

Figure 4.31 shows 100 trajectories of the flat agent, performed after the 2250^{th} learning episode, with a greedy policy with respect to Q_{flat}. The room environment and the model realization are the same as those of Figure 4.28 and Figure 4.29. With respect to Figure 4.28, the agent performs shorter trajectories, heading directly for the goal. This is not surprising, since the flat agent is already trained in that same room environment. However, several collisions can be individuated in the trajectories: out of 100 trajectories, only 13 reach the goal. The average number of collisions per time-step is equal to $1.28 \cdot 10^{-2}$, which is sensibly higher than its SHRL equivalent.

The higher number of collisions caused by the flat controller has two explanations. First of all, the use of a tabular Q-function Q_{flat}, whose refinement might not be sufficient for such a task. While the use of a function approximator might reduce this problem and make the function more efficient, it should be observed that the discretization of Q_{flat} is still more refined than that of $\mathcal{P}^1$ adopted to learn π_{SubM}, and than that of x_1, x_2 adopted to learn π_{SupM}. Furthermore, it can be seen from Figure 4.31 that the MAV collides often with the obstacles surrounding the goal. This can be interpreted as a proof that the conflict between Q_{exp}, which learns to direct the MAV towards the goal, and Q_{obs}, which tends to avoid obstacles, is not correctly addressed with the proposed flat implementation. Conversely, the SHRL agent, due to its hierarchical structure, prevents this conflict by enforcing safety on the MOS *SubM*.

Summarizing, the VST policy π_{V} is not able to entirely avoid obstacles, as it is the case in Section 4.4, due to the limitations in terms of belief set, and to the inaccuracies between the models used during training. However, the obtained trajectories show that the agent adopts a cautious behavior, avoiding the most cluttered areas of the room, and is ultimately able to discover the position of the goal within the unknown, continuous environment. Furthermore, a flat RL agent, allowed to learn for an extensive number of episodes within the "real-world" environment, and including the actual model realization, is evaluated in the same task and is found to have a rate of collisions per step that is considerably higher than the VST-trained SHRL agent.

4.6. Conclusions

This chapter focuses on the use of Hierarchical Reinforcement Learning (HRL) to increase safety of exploration with respect to flat RL. The concept of temporally extended actions (TEAs) is discussed, together with a brief description of the most common formalisms of HRL in the literature: Hierarchy of Abstract Machines (HAM), options, and MAXQ. After briefly discussing the basic properties of HRL, their application to safe exploration is formalized as a new branch denominated Safe Hierarchical Reinforcement Learning (SHRL).

Afterward, the use of HRL methods for safety is exemplified by one specific approach, i.e., Virtual Safety Training (VST). The state is projected through a projection function, and a hierarchical architecture is adopted using this projection to learn the policy of the machine, option or subtask (MOS) that executes primitive actions. Given certain assumptions, these policies can guarantee or increase safety of exploration.

This approach is tested on two different scenarios. The first one is a maze exploration task, which constitutes an example of an SRL problem with deterministic dynamics and in a discrete environment. These simplifications result in a VST that guarantees safety. The final SHRL policy, obtained by reiterating the low-level VST policy on multiple levels of state abstraction, is not only safe, but it is shown to find the goal in a lower amount of time-steps than similar algorithms.

In order to provide a more realistic example, VST is applied to an MAV task. The MAV objective is to discover a goal position inside a 2D room environment which contains obstacles. The position and shape of the obstacles, as well as the exact dynamics of the MAV, are unknown. In order to accommodate these aspects, the VST procedure includes state abstraction for the definition of the agent's beliefs. The results show that the VST-trained SHRL agent outperforms a more extensively trained flat RL agent, reducing the number of collisions.

In conclusion, with respect to the other methods, such as graph pruning, the introduction of hierarchical structures allows the designer to define agents that can explore environments that are more uncertain and less structured. Therefore, it can be argued that using SHRL methods is an effective way of increasing the robustness of the agent with respect to flat RL.

II

HYBRID METHODS

5

Safety metrics

*This chapter addresses the challenges of **safety** and **online efficiency**, by incorporating the graph representation of the bounding model dynamics of Chapter 3 into the backup formulation of Chapter 2. It also defines two graph-based **safety metrics**: the operational metric and the proximity metric. These are used to assign weights to the vertices of the graph, which in turn permits to assess the safety of the agent's actions with a finite horizon optimization akin to that of OptiSHERPA of Chapter 2. The two metrics are shown to lead to varying degrees of success in preventing fatal occurrences, depending on the complexity of the environment.*

5.1. Introduction

Chapter 2 presented OptiSHERPA, a heuristic method that optimizes a distance or evasion metric, depending on the current state, on the known safe state space, and on a list of previously visited states. This use of the metrics restricts the number of admissible control sequences when searching for backups, and allows the agent to classify actions in order of safety.

However, even though the number of admissible sequences is reduced, several interval computations are needed in order to generate backups, which is undesired for UAVs in accordance with the *challenge of efficiency*. This chapter shows how a reduced computational burden for backups can be achieved by adopting *safety metrics*, defined in terms of the graph representation of the environment dynamics, which was introduced in Chapter 3 as a solution to the challenge of efficiency. The metrics are tested on two applications in order to validate the metrics in different environments.

The chapter is structured as follows. Section 5.2 summarizes the assumptions and the procedures for the application of the safety metrics, which are illustrated in detail in Section 5.3. The optimization algorithm is then introduced in Section 5.4 and applied to two diverse tasks in Section 5.5. Section 5.6 concludes.

5.2. Assumptions and graph generation

This section briefly provides the assumptions on the environment and on the model necessary for using the safety metrics, as well as the procedure to generate the graph. With respect to Chapter 3, this section considers environments with both continuous and discrete states.

5.2.1. Assumptions

Let $\boldsymbol{x} \in \mathcal{S}$ be the *hybrid* state of the environment in the form:

$$\boldsymbol{x} = (c_1, \ldots, c_m, d_1, \ldots, d_n) \tag{5.1}$$

with the generic continuous component $c_i \in [\underline{c}_i, \overline{c}_i] \subset \mathbb{R}$ and the discrete components $d_j \in D_j$, where $D_j \subset \mathbb{R}$ is an ordered set. Let $\mathcal{A}$ be the time-state-independent set of available actions u of the RL agent. The agent's behavior is described by policy $\pi : \mathcal{S} \times \mathcal{A} \to [0, 1]$. Let $\mathcal{D}$ be the dynamics of the environment, which are assumed to be a system of differential and/or difference equations as indicated by

$$\mathcal{D} : \begin{cases} \dot{c}_1 & = \mathcal{D}_1\,(c_1, \ldots, c_m, d_1, \ldots, d_n, u) \\ & \vdots \\ \dot{c}_m & = \mathcal{D}_m\,(c_1, \ldots, c_m, d_1, \ldots, d_n, u) \\ \Delta d_1 & = \mathcal{D}_{m+1}\,(c_1, \ldots, c_m, d_1, \ldots, d_n, u) \\ & \vdots \\ \Delta d_n & = \mathcal{D}_{m+n}\,(c_1, \ldots, c_m, d_1, \ldots, d_n, u). \end{cases} \tag{5.2}$$

As usual, the dynamics $\mathcal{D}$ and the reward function $\mathcal{R} : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \rightarrow \mathbb{R}$ are not explicitly known by the agent. However, it is be assumed that the agent possesses a known bounding model of the form

$$\hat{\mathcal{D}} : \begin{cases} \dot{c}_1 & \in \hat{\mathcal{D}}_1 \left(c_1, \dots , c_m, d_1, \dots , d_n, u\right) \\ & \vdots \\ \dot{c}_m & \in \hat{\mathcal{D}}_m \left(c_1, \dots , c_m, d_1, \dots , d_n, u\right) \\ \Delta d_1 & \in \hat{\mathcal{D}}_{m+1} \left(c_1, \dots , c_m, d_1, \dots , d_n, u\right) \\ & \vdots \\ \Delta d_n & \in \hat{\mathcal{D}}_{m+n} \left(c_1, \dots , c_m, d_1, \dots , d_n, u\right). \end{cases} \tag{5.3}$$

Let $\mathrm{FSS} \subset \mathcal{S}$ be the unknown set of fatal states which the agent must avoid, and let

$$W(\boldsymbol{x}) = \begin{cases} 0 \text{ if} N(\boldsymbol{x}) \cap \mathrm{FSS} = \emptyset \\ 1 \text{ else} \end{cases} \tag{5.4}$$

be the *warning* function, where N is the set of those states $\boldsymbol{x}' = (c_1', \dots, c_m', d_1', \dots, d_n')$ which constitute a neighborhood of $\boldsymbol{x}$ given fixed widths n^{c_i} and n^{d_j}:

$$N(\boldsymbol{x}) = \left\{\boldsymbol{x}' = (c_1', \dots, c_m',\ d_1', \dots, d_n') \mid \left\|c_i - c_i'\right\| < n^{c_i},\ \left\|d_j - d_j'\right\| < n^{d_j}\right\}, \tag{5.5}$$

where $\|*\|$ is the absolute value of $*$, $1 \le i \le m$, $1 \le j \le n$. Since the FSS is unknown, W is not a-priori known: the agent receives the output of this function only during exploration. Assuming that the FSS is constant with time, the agent can thus estimate the fatal set during exploration by looking at the output of W, which constitutes risk perception.

5.2.2. Graph generation

This subsection summarizes how to obtain a graph representation of the hybrid dynamics of Eq. (5.3), as shown in Section 3.2.3. The state space $\mathcal{S}$ is discretized into a tiling $\mathcal{T}$. The continuous components of $\boldsymbol{x}$ are partitioned evenly as

$$[\underline{c}_j,\ \overline{c}_j] = [\underline{c}_j,\ \underline{c}_j + \Delta_j] \cup \dots \cup [\overline{c}_j - \Delta_j,\ \overline{c}_j]. \tag{5.6}$$

This results in a continuous tiling of $\prod_{j=1}^{n} \frac{\overline{c}_j - \underline{c}_j}{\Delta_j}$ elements. If the system includes discrete components, this tiling does not yet entirely cover the state space. One copy of the continuous tiling is generated for each combination of discrete components. Therefore, the complete tiling $\mathcal{T}$ numbers

$$\prod_{i=1}^{n} \prod_{j=1}^{m} \frac{\overline{c}_i - \underline{c}_i}{\Delta_i} \left|D_j\right| \tag{5.7}$$

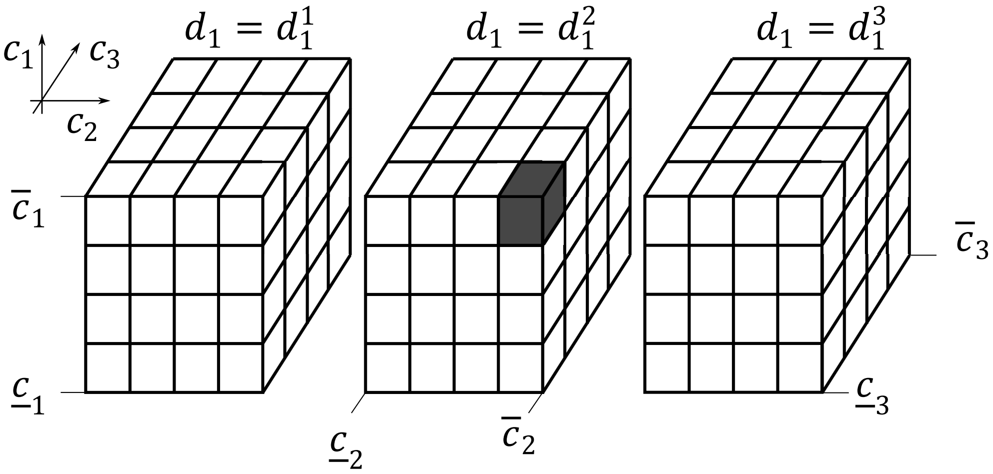


Figure 5.1: A simple example of an even tiling of an hybrid system with 3 continuous components ($c_1 \in [\underline{c}_1, \overline{c}_1]$, $c_2 \in [\underline{c}_2, \overline{c}_2]$, $c_3 \in [\underline{c}_3, \overline{c}_3]$) and one discrete component $d_1 \in D_1 = \{d_1^1, d_1^2, d_1^3\}$. Each continuous component is partitioned in four intervals, for a total of $4 \cdot 4 \cdot 4 \cdot 3 = 192$ tiles.

tiles τ, where $|*|$ is the number of elements in $*$. Each tile represents an interval of values c_i in the continuous components and a combination of values d_j of discrete ones. An example of tiling is given in Figure 5.1. Each tile τ and its elements are uniquely indicated by the index $\mathtt{i}(\tau)$, as mentioned in Section 3.2.3. For example, the shaded tile of Figure 5.1 has index $\mathtt{i}(\tau) = (4, 4, 1, 2)$ and is equivalent to the set

$$\begin{aligned}\{\, \boldsymbol{x} = (c_1, c_2, c_3, d_1) \mid c_1 \in [\overline{c}_1 - \Delta_1, \overline{c}_1]\,, \\ c_2 \in [\overline{c}_2 - \Delta_2, \overline{c}_2]\,,\ c_3 \in \left[\underline{c}_3, \underline{c}_3 + \Delta_3\right],\ d_1 = d_1^2 \,\}. \quad (5.8)\end{aligned}$$

As a second step, an arbitrary subset $\mathcal{A}_{sub}$ is extracted from the action set $\mathcal{A}$ of which $\mathcal{A}_{sub}$ must be representative.

As a final step, the bounding model of Eq. (5.3) must be discretized in time. Given a timestep Δt, it is straightforward to obtain a time-discrete equivalent of the continuous dynamics of Eq. (5.3). The final formulation of the dynamics[1] is then

[1] It will be assumed for simplicity that the timestep Δt is a multiple of the sampling time δt of the discrete transitions of components $d_j(k)$, so that $\Delta d_j = d_j(k + \frac{\Delta t}{\delta t}) - d_j(k)$, $\frac{\Delta t}{\delta t} \in \mathbb{N}$.

$$\hat{\mathcal{D}}^{\Delta t} : \begin{cases} \Delta c_1 \in \hat{\mathcal{D}}_1 \left(c_1, \cdots, c_m, d_1, \cdots, d_n, u \in \mathcal{A}_{sub}, \Delta t\right) \\ \vdots \\ \Delta c_m \in \hat{\mathcal{D}}_m \left(c_1, \cdots, c_m, d_1, \cdots, d_n, u \in \mathcal{A}_{sub}, \Delta t\right) \\ \Delta d_1 \in \hat{\mathcal{D}}_{m+1} \left(c_1, \cdots, c_m, d_1, \cdots, d_n, u \in \mathcal{A}_{sub}, \Delta t\right) \\ \vdots \\ \Delta d_n \in \hat{\mathcal{D}}_{m+n} \left(c_1, \cdots, c_m, d_1(k), \cdots, d_n(k), u \in \mathcal{A}_{sub}, \Delta t\right). \end{cases} \tag{5.9}$$

Given the above formulation, the graph $\mathcal{G}$ can finally be generated with the same procedure introduced in Section 3.2.4.

5.3. Metrics

This section introduces two metrics: an *operative metric* (OM) that directly represents the information deriving from the warning function, and a *proximity metric* (PM) that accounts for the degree of exploration of the system in near-time. Both metrics are used by the agent to assign weights to the vertices of the graph $\mathcal{G}$. Given the weights, at each time-step the agent selects the safest action available by solving a finite horizon optimization, akin to model predictive control (MPC) [121].

Each metric is extended to *collections* of vertices C. A collection is defined as a subset of $\mathcal{T}$ so that, if τ_1 with index $\mathtt{i}(\tau_1) = (\mathtt{i}_1^1, \ldots, \mathtt{i}_{m+n}^1)$ and τ_2 with index $\mathtt{i}(\tau_2) = (\mathtt{i}_1^2, \ldots, \mathtt{i}_{m+n}^2)$ are both elements of C, all vertices τ for which

$$\mathtt{i}_j^1 \leq \mathtt{i}_j(\tau) \leq \mathtt{i}_j^2, \ \forall j \in \{1, \ldots, m+n\} \tag{5.10}$$

are also elements of C. A collection is a connected and convex set of tiles, with respect to both continuous and discrete components of the state. It is worth mentioning that, since the dynamics of Eq. (5.9) yield intervals, the graph $\mathcal{G}$ yields collections of tiles.

Operative metric

The output of $W(\boldsymbol{x})$ is used by the agent to avoid unsafe states. The goal of the OM is to embed the information obtained by W as weights $q(\tau)$ of vertices τ that can approximate the $\mathtt{FSS}$ and that can be efficiently modified or recalled 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 weight equal to q_{unc} representing the notion that $\mathtt{FSS}$ is unknown at the start. At the start of each timestep, the warning function is evaluated. Supposing that no risk is perceived, i.e., $W(\boldsymbol{x}) = 0$, all tiles τ that entirely fall within the neighborhood $N(\boldsymbol{x})$ (Eq. (5.5)) contain only safe states. The weight of the corresponding vertices $q(\tau)$ is then increased to q_{safe}, unless the weight is already higher. Conversely, those tiles that are only partially in range, or not in range at all, might contain fatal states: the weights of the corresponding vertices are then not altered. Additionally, the weight of the current vertex corresponding to the currently occupied tile is increased to

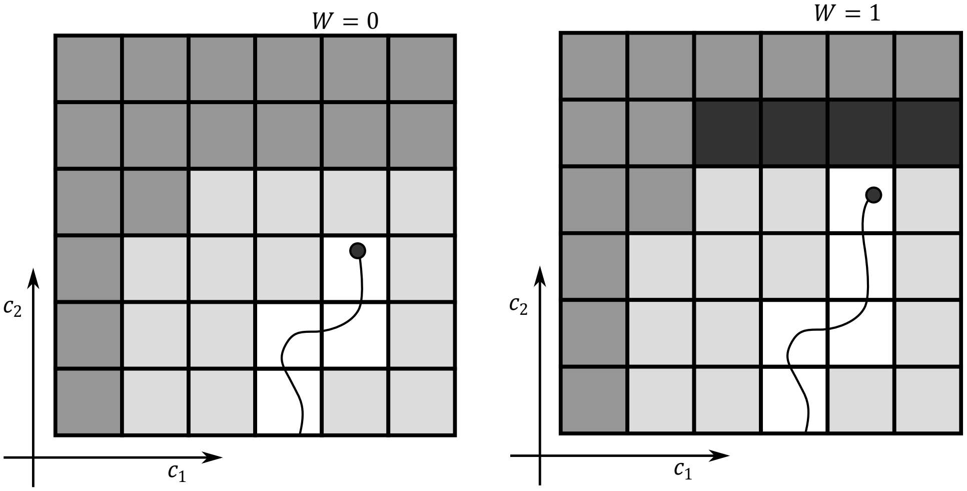


Figure 5.2: An example of update of weights for the OM. In the left figure, explored tiles, safe tiles and uncertain tiles are indicated with increasingly darker color. No warning ($W = 0$) is perceived in this state (represented by the black dot), but in the next state (black dot on the right) a warning is triggered $W = 1$. The uncertain tiles now in range of risk perception (in black) are then labeled as fatal.

q_{exp}. This is done to include in the metric information about previously visited areas of $\mathcal{S}$, in order to partially counteract the presence of LTF states (Eq. (2.3)).

However, if $W = 1$, at least one of the states currently in range of risk perception is a fatal state. In order to be conservative, and guarantee an overapproximation of the FSS, the weight of all vertices corresponding to unexplored tiles (i.e., which have weight equal to q_{unc}) that are at least partially in range are lowered to q_{fat}. By adopting the above passages, all reliable information from the warning function is transferred into the graph weights. Figure 5.2 provides an example of the application of the OM. To conclude, the OM is extended to a collection C by averaging the value $q(\tau)$ of all τ in the collection:

$$q(C) = \frac{\sum_{\tau \in C} q(\tau)}{|C|}. \tag{5.11}$$

In this case, value $q(C)$ indicates the average estimated danger of entering a state in the collection.

Proximity metric

While the OM assigns weights primarily depending on the perceived risk, the proximity metric (PM) is an equivalent of the distance metric of Section 2.4.2, when considering tiles instead of states. The weights are therefore assigned depending on distances to previously explored states.

Consider the continuous components of two states $\boldsymbol{x} \in \tau$ and $\boldsymbol{x}' \in \tau'$, with $\mathrm{i}(\tau) = (\mathrm{i}_1, \dots, \mathrm{i}_{m+n})$, $\mathrm{i}(\tau') = (\mathrm{i}'_1, \dots, \mathrm{i}'_{m+n})$. From the definition of index, the more two values of the same component differ, the more the difference in index:

$$c_i \ll (\gg)\, c_i' \rightarrow \mathtt{i}_i \ll (\gg)\, \mathtt{i}_{i'}', \tag{5.12}$$

and conversely,

$$c_i \approx c_i' \rightarrow \|\mathtt{i}_i - \mathtt{i}_i'\| \leq 1. \tag{5.13}$$

Therefore, indexes can be used to indicate distances instead of the actual values of the continuous components. Concerning the discrete components d_j of the state, it will be assumed that the absolute difference $\|d_j - d_j'\|$ is equivalent to a distance between two continuous component $\|c_j - c_j'\|, c_j = d_j,\ c_j' = d_j'$.

Given the above considerations, the *distance* $\mathrm{dist}(\tau, \tau')$ between two tiles τ and τ' is defined as

$$\mathrm{dist}(\tau, \tau') = \left\|\boldsymbol{v}_\mathrm{r} \odot (\mathtt{i}(\tau) - \mathtt{i}(\tau)')\right\|_2, \tag{5.14}$$

where $\| * \|_2$ indicates the Euclidean norm of $*$, $\odot$ indicates element-by-element product and $\boldsymbol{v}_\mathrm{r} \in \mathbb{R}^{n+m}$ is a rescaling vector of positive gains (see Section 2.4.2). 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, this means that the ordering and value of the discrete components d_j allows to compare differences in values d_j to differences in the values of continuous components c_i, in terms of safety and controllability of the environment. The term $\boldsymbol{v}_\mathrm{r}$ acts as a rescaling vector for the 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. More relevant components of the state should 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 anytime that a new tile is visited, the agent appends it to a list of explored tiles $\mathcal{T}_\mathrm{list}$. Given this list, the *proximity* of a vertex/tile τ is

$$\mathrm{prox}(\tau) = -\min_{\tau' \in \mathcal{T}_\mathrm{list}} \mathrm{dist}(\tau,\ \tau'), \tag{5.15}$$

which is the distance to the nearest tile containing an explored state, changed in sign. It can be seen that, since the metric depends on $\mathcal{T}_\mathrm{list}$, weights must be recomputed when the list is updated.

The following extension is applied when considering a collection of states C. First, the center τ_C of the collection is found. Then, the proximity of C is equal to the proximity of τ_C minus an additional uncertainty term:

$$\mathrm{prox}(C) = \mathrm{prox}(\tau_C) - \rho \max_{\tau' \in C} \mathrm{dist}(\tau_C, \tau') \tag{5.16}$$

weighted with $\rho < 1$. This additional term is proportional to the distance of τ_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

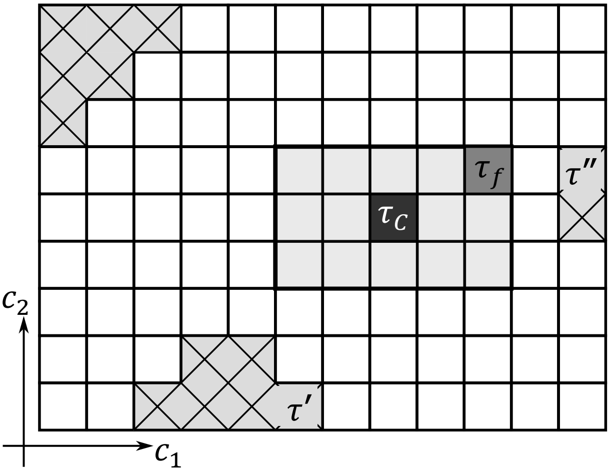


Figure 5.3: An example of proximity computation for collection C. The black tile τ_C is the center of the collection. The tiles indicated by a cross contain a visited state. Among these, with the assigned weight vector $\boldsymbol{v}_\mathrm{r} = (2, 1)$, τ' is the nearest tile to τ_C. A term proportional to the distance between τ_C and τ_f is then added to compute the proximity.

dispersion in the collection itself, equivalently to the distance metric of Section 2.4.2. Figure 5.3 shows an example of an application of the metric. The state space has two continuous components, c_1 and c_2. The gray squares represent the collection C, with center τ_C. The cross marked tiles represent those $\tau \in \mathcal{T}_\mathrm{list}$ containing a visited state. With a weight vector $\boldsymbol{v}_\mathrm{r} = (2, 1)$, tile τ' is the one with the lowest distance of $-4\sqrt{2}$, higher than that of τ'', which is equal to $\sqrt{65}$. Therefore τ' is the nearest tile under this metric even though τ'' is closer in index within the tiling. Finally, a term proportional to the distance between τ_C and τ_f, the furthest tile of the collection, is added to obtain the proximity of the whole collection.

5.4. Algorithm description

This section illustrates the algorithm for safe exploration which, as OptiSHERPA of Section 2.4.3, consists in solving an optimization problem at each time-step. It is assumed that in its initial state $\boldsymbol{x}_0$, $W(\boldsymbol{x}_0) = 0$, i.e., the agent does not perceive risk. The goal of the algorithm is to indicate which action among the available set $\mathcal{A}_{sub}$ is the safest. In particular, *commands* $\boldsymbol{a}$ of length k and of the form

$$\boldsymbol{a} = (u(t), \cdots, u(t + (k-1) \cdot \Delta t)) \tag{5.17}$$

are considered. The graph $\mathcal{G}$ is invoked to predict the collection of tiles $C(\boldsymbol{a})$ overestimating the state of the system after the application of command $\boldsymbol{a}$. The resulting $C(\boldsymbol{a})$ is then evaluated with a safety metric. The safest command $\boldsymbol{a}^*$ is the one for which $C(\boldsymbol{a}^*)$ maximizes a metric of choice.

In practice, rather than solving the problem in closed form, a finite set of candidate commands A is examined, based on the following restrictions. A first one involves the length of the command. As explained in Section 3.2.4, due to the over-approximating nature of $\mathcal{G}$, there is a limit on how many steps ahead can efficiently be predicted, with collections increasing in size, and being less and less predictive with the increase in time-steps. Candidate commands are then restricted in length so that $k \leq k_{max}$. A lower constraint on the length of the command is also imposed: $1 \leq k_{min} \leq k$. A final selection is performed by considering candidates of the form

$$u(t) = u(t + \Delta t) = \cdots = u(t + (k-1) \cdot \Delta t), \tag{5.18}$$

i.e., constant commands. The reason for this choice comes from the previous assumption that the action set $\mathcal{A}_{sub}$ is representative of the entire action set $\mathcal{A}$. Due to the limited amount of time-steps that can be accounted for via the graphical representation of the dynamics, it is then more meaningful to consider constant commands that truly represent the effect of the individual actions, rather than commands with mixed and possibly conflicting control actions. Additionally, this choice drastically reduces the number of candidates to $|A| = |\mathcal{A}_{sub}| \cdot (k_{max} - k_{min} + 1)$.

Figure 5.4 summarizes the algorithm. The composing elements of the algorithm are the graph $\mathcal{G}$, a predefined set of action $\mathcal{A}_{sub}$ generating the set of candidates commands A, a warning function $W(\boldsymbol{x})$ and an arbitrary safety metric. The system is in state $\boldsymbol{x}(t)$, which in the graph $\mathcal{G}$ corresponds to current vertex τ. By using the adjacency matrices of $\mathcal{G}$ the trajectory of the system under command $\boldsymbol{a} \in A$ can be obtained, as well as the final collection of states $C(\boldsymbol{a})$. Collections C are evaluated by the metric for all candidate commands. The optimal command $\boldsymbol{a}^*$ is selected, and its first action $u^*(t)$ is performed in the system. The new state $\boldsymbol{x}(t+\Delta t)$ is then observed, and the metric is updated with the information derived from exploration and from W. The process then repeats.

In addition to the above restrictions, two more checks are performed during optimization. As indicated by the graph generation procedure, $\mathcal{G}$ does not contain outbound edges for which $\boldsymbol{x}(t + \Delta t) \notin \mathcal{S}$. If the control action indicated by the current command is one such action, it is temporarily removed from the set of candidates during that optimization process. Additionally, it is important for the validity of the method to verify that all vertices τ in the trajectory are safe, according to the current knowledge of the FSS. Commands for which this cannot be guaranteed are also temporarily removed from A.

5.5. Applications

This section presents two simulated applications of the algorithm of Section 5.4. The goal of the first task is to fly a quadrotor within a room without colliding with its walls. The objective of the second task is to control an aircraft flying with a constraint on its altitude via elevator deflection. The RL agents assigned to these tasks, which have very different dynamics and therefore represent different challenges, are equipped with either the OM or the PM.

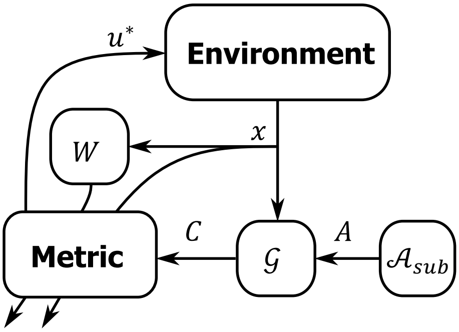


Figure 5.4: The algorithm for action selection. Given current state $\boldsymbol{x}$, the corresponding tile in graph $\mathcal{G}$ is found. Candidate commands A yield a set of collections C, which is evaluated under the current metric, to find the optimal action u^*. In addition, the metric is updated with the visited state and with information from the warning function W.

5.5.1. Quadrotor navigation task

The goal of this simulated task is to fly a quadrotor for a set amount of time-steps without colliding with the surrounding walls. The dynamics $\mathcal{D}$ of the quadrotor are represented by the hybrid system of Eq. (5.19):

$$\begin{gathered}\dot{x}_1 = V\ \cos(\psi);\quad \dot{x}_2 = V\ \sin(\psi);\quad \dot{V} = \theta\ \dot{V}_c;\\ \dot{\psi} = \begin{cases} +\dot{\psi}_c & \text{if}\,\texttt{right}\\ -\dot{\psi}_c & \text{if}\,\texttt{left}\\ 0 & \text{if}\,\texttt{neut}\end{cases};\ \Delta\theta = \begin{cases} +1 & \text{if}\,\texttt{forw} \wedge \theta \neq +1\\ -1 & \text{if}\,\texttt{back} \wedge \theta \neq -1\\ 0 & \text{if}\,\texttt{neut}.\end{cases}\end{gathered} \tag{5.19}$$

The state $\boldsymbol{x} = (x_1, x_2, V, \psi, \theta)$ contains the absolute position (x_1, x_2), the speed V, and the heading ψ of the quadrotor, while θ is a discrete component which represents the pitch configuration: forward ($\theta = 1$), backward ($\theta = -1$), or level ($\theta = 0$). The finite set of actions $\mathcal{A}$ comprises `forw` and `back` to alter the pitch, `right` and `left` to steer, and the neutral action `neut`. Finally, the dynamics of Eq. (5.19) depend on the values of two additional parameters: the mean acceleration $\dot{V}_c$ and the mean turning rate $\dot{\psi}_c$, which specify the quadrotor performance. In accordance with the assumptions of Section 5.2.1, only the intervals $\hat{\dot{V}}_c = [0.24,\ 0.6]\frac{\text{m}}{\text{s}^2}$ and $\hat{\dot{\psi}}_c = [\pi/4,\ \pi/3]\ \text{s}^{-1}$ of these values are known to the agent. The uncertain bounding model $\hat{\mathcal{D}}$ is then obtained by replacing $\dot{V}_c$ and $\dot{\psi}_c$ in Eq. (5.19) with these intervals.

The graph $\mathcal{G}$ is generated as follows. First, the unbounded state space $\mathcal{S}$ is restricted so that it conforms to the assumption of Eq. (5.1). Components $c_1 = x_1$ and $c_2 = x_2$ are physically bounded by the size of the square room: $x_1, x_2 \in [-5,\ 5]$ m. The angle $\psi = c_4$ is bounded between $-\pi$ and π, and $d_1 = \theta$ is already restricted in the formulation of the dynamics. Therefore, only the speed $V = c_3$ needs to be artificially restricted. In the present work it is $V \in [-1.2,\ 1.2]\frac{\text{m}}{\text{s}}$. Although any arbitrary value could be selected, it was observed in simulations that this choice of V results in a good trade-off between exploration and maneuverability. The continuous components are then evenly divided into 20 intervals, resulting in a finite tiling of $4.8 \cdot 10^5$ tiles. The action set $\mathcal{A} = \{\texttt{forw}, \texttt{back}, \texttt{right}, \texttt{left}, \texttt{neut}\}$ is already finite, so $\mathcal{A}_{sub} = \mathcal{A}$ is chosen. Finally, a time-step Δt of $0.5s$ is adopted to generate the graph. This choice of Δt is motivated by the need of a sufficiently long time-step to correctly represent the pitch configuration transition.

The function W, which simulates the presence of on-board sensors, outputs a warning to the agent when the quadrotor is within $2.5\mathrm{m}$ of a wall. For action selection, candidate commands $\boldsymbol{a}$ are chosen as constant commands with duration comprised between $k_{min} = 3$ and $k_{max} = 5$ time-steps. The task fails if the quadrotor hits a wall, or if otherwise $\boldsymbol{x} \notin \mathcal{S}$, before the 300^{th} time-step. Each episode is initialized in a random state

$$x_1|_0 = 0\,;\; x_2|_0 = 0\,;\; V|_0 \in [0.4,\ 0.6]\,;\; \psi|_0 \in [-\pi,\ \pi]\,;\; \theta_0 = 0 \tag{5.20}$$

and the actual model of the quadrotor is randomly assigned two values

$$\dot{V}_c \in \hat{\dot{V}}_c\,;\; \dot{\psi}_c \in \hat{\dot{\psi}}_c \tag{5.21}$$

among all the possible realizations of the dynamics represented by Eq. (5.19). In the results to follow, three agents are compared, selecting actions according to the OM, according to the PM, or according to a random policy.

Operative metric for quadrotor control

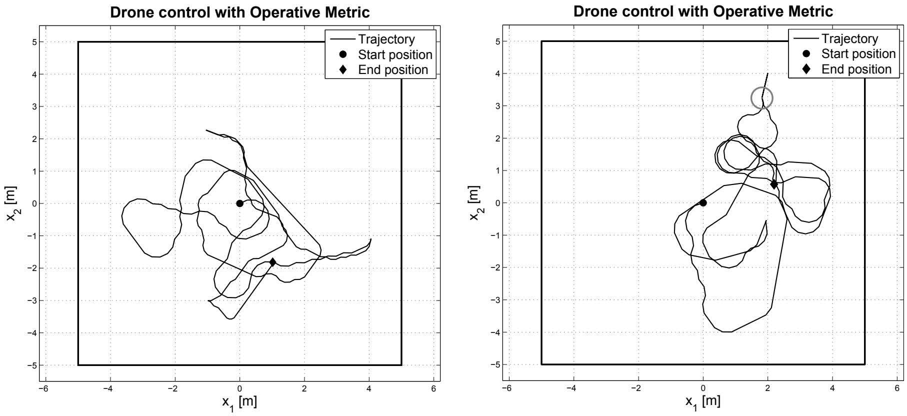
Parameters q are initialized as $q_{exp} = 1$, $q_{safe} = 0$, $q_{unc} = -100$, $q_{fat} = -10^6$: it can be seen how the above initialization heavily penalizes uncertain and possibly fatal tiles with respect to safe or explored tiles. A typical behavior resulting from the application of the metric is shown in Figure 5.5. In the first instants of flight, the quadrotor is far from the walls; the agent does not alter the pitch configuration, but instead selects actions `neut`, `right` and `left` repeatedly to move around the room at constant speed. After a few iterations, when the central region of the room is 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 the UAV is in proximity of a wall, the agent adopts one of the following two behaviors. The first behavior is simply to steer with a constant rate until the collision is avoided. The second one, represented in Figure 5.5b, is to pitch backward, reduce the speed, and eventually invert the direction of flight. Simulations show how applying the OM results in a safe flight that avoids collisions and at the same time explores the environment accordingly.

Proximity metric for quadrotor control

The rescaling vector $\boldsymbol{v}_\mathrm{r}$ is selected as

$$\boldsymbol{v}_\mathrm{r} \;=\; (v_\mathrm{r}|_{x_1}\; v_\mathrm{r}|_{x_2}\; v_\mathrm{r}|_{V}\; v_\mathrm{r}|_{\psi}\; v_\mathrm{r}|_{\theta}) \;=\; (5,\ 5,\ 2,\ 1,\ 1) \tag{5.22}$$

in accordance with the principle that lower weights should be assigned to those components of the state that are immediately accessible from the controller, in this



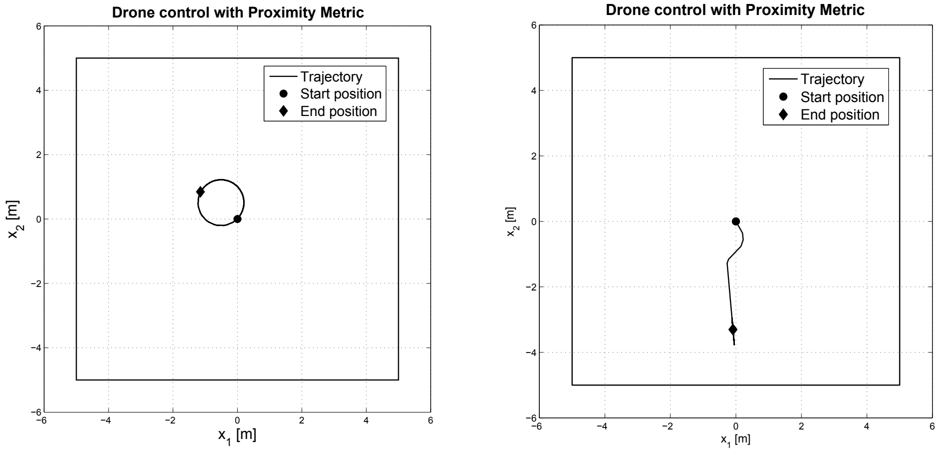
(a)

(b)

Figure 5.5: Two sample trajectories with the operative metric. The dot represents the starting position of the quadrotor, while the diamond is the end of the trajectory. The bold lines delimit the room. The trajectory in the left figure (a) shows both inversions of pitch and turns performed at constant speed. In the right figure (b) an inversion of direction of flight (indicated by a circle) is performed in order to avoid a collision.

case ψ and θ. Increasingly higher gains are assigned to V and to x_1 and x_2, so that the agent is more cautious with respect to alteration in those components, which are harder to control. A value of 0.3 is given to the parameter weight ρ of Eq. (5.16). The results show two different behaviors depending on the initial trajectory of the system. The size of the room and the turning rate $\dot{\psi}_c$ are sufficient for the UAV to make a complete turn if the agent applies a constant `right` or `left` action. The tiles containing positions x and y during this maneuver are then added to the list $\mathcal{T}_{\text{list}}$ of visited tiles, together with the corresponding heading. The PM indicates that the command for which the collection C is the closest is to keep turning. As a result, the UAV keeps turning indefinitely, see Figure 5.6a. This behavior and the resulting constant maneuver are safe; however, the agent does not explore its environment further. This is the result of the concept behind the metric, with the agent trying to replicate already encountered conditions.

A different behavior can be observed in Figure 5.6b. The agent pitches backward, reducing the flight speed almost to zero, and then selects repeatedly the neutral action `neut`. The quadrotor then drifts with constant speed and heading, until it comes in the proximity of a wall. The agent then pitches backward in order to brake. Depending on the initial speed V_0 and on the acceleration $\dot{V}_c$, this is likely to result in a collision. However, if the UAV does not collide, it drifts in the opposite direction, with the agent adopting the `neut` action. As before, the tiles τ which contain the position, heading, and speed during these drifts are added to $\mathcal{T}_{\text{list}}$. Therefore, the PM instructs the agent to pitch forward again as soon as the UAV approaches the initial drifting position. From this point onward, the UAV repeats these two drifting trajectories.



(a)

(b)

Figure 5.6: The two maneuvers exhibited by the agent adopting the proximity metric. In the left figure (a) the agent enters a safe maneuver by performing a constant turn. On the right (b) the quadrotor pitches forward and backward until the end of the trial. In both cases, very limited exploration is achieved.

Comparing the different agents, the mean duration of the task observed with a random policy is of 19.3 iterations, equivalent to $9.65\mathrm{s}$. The agent equipped with the OM achieves completion of the task at every run. With the PM, the agent completes the task on 44% of the runs, with minimum duration of 34 iterations and an average duration of 161.

5.5.2. Elevator control task

In this task, the agents controls the elevator δ_e of an aircraft to prevent it from stalling and from leaving an initially unknown altitude range for a set number of time-steps. The nominal longitudinal dynamics of the aircraft are

$$\begin{pmatrix} \dot{h} \\ \dot{\theta} \\ \dot{\alpha} \\ \dot{q} \end{pmatrix} = A \begin{pmatrix} h \\ \theta \\ \alpha \\ q \end{pmatrix} + B_\mathrm{n}\delta_e;\ A = \begin{bmatrix} 0 & 300 & -300 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & -0.64 & 0.938 \\ 0 & 0 & -1.568 & -0.879 \end{bmatrix};\ B_\mathrm{n} = \begin{bmatrix} 0 \\ 0 \\ B^{\alpha}_{\delta_e} \\ B^{q}_{\delta_e} \end{bmatrix} \tag{5.23}$$

where the speed V is assumed constant and equal to $300\frac{\mathrm{ft}}{\mathrm{s}}$, and where

$$B^{\alpha}_{\delta_e} = -1.4 \cdot 10^{-3}\,;\ B^{q}_{\delta_e} = -0.1137. \tag{5.24}$$

The state of the aircraft is $\boldsymbol{x} = (h, \theta, \alpha, q)$, where $h[\mathrm{ft}]$ is the deviation from initial altitude, $\theta[\mathrm{rad}]$ is the pitch angle, $\alpha[\mathrm{rad}]$ is the angle of attack, and $q[\mathrm{rad\,s^{-1}}]$ is the pitch rate. The agent action u is the elevator deflection $\delta_e \in [-4°, 4°] = \mathcal{A}$.

It is assumed that the actual dynamics of the aircraft differ from the nominal model, e.g., due to deteriorations of the control surfaces or to inaccuracies in the model identification. However, a bounding model is provided to the agent by replacing the nominal control matrix B_n with

$$\hat{B} = [1.05 \cdot B_\mathrm{n},\ 0.95 \cdot B_\mathrm{n}]. \tag{5.25}$$

The goal of the agent is to prevent the aircraft from leaving an altitude range of $[-80,\ 80]\mathrm{ft}$ from the initial altitude. Additionally, a second risk is represented by a stall, which for simplicity is assumed occurring outside of a range $\alpha \in [-15°,\ 12°]$. Neither ranges of h or α are initially known to the agent; however, a function W issues a warning when $\boldsymbol{x}$ reaches within $30\mathrm{ft}$ or $\frac{\pi}{30} = 6°$ of the boundaries of the above ranges.

From Eq. (5.23) it can be seen that the main effect of control action δ_e is a pitch acceleration $\dot{q}$. The dynamic of α is loosely affected by the instantaneous deflection δ_e due to the relatively small coefficient $B^{\alpha}_{\delta_e}$, but is instead dominated by the values of q and α. Furthermore, the altitude rate $\dot{h}$ depends on the angle $\gamma = \theta - \alpha$, whose derivative can be written as

$$\dot{\gamma} = \dot{\theta} - \dot{\alpha} \cong 0.062 \cdot q + 0.64 \cdot \alpha. \tag{5.26}$$

In conclusion, $\dot{h}$ depends on γ, which is mainly controlled through α. In turn, α is dominated by q, which is controllable through elevator deflection δ_e. Therefore, this task is an example of low-level control in the presence of cascaded dynamics.

While h and α are bounded by these ranges, θ and q are arbitrarily limited respectively to $[-\pi/4,\ \pi/4]$ and $[-\pi/2,\ \pi/2]\mathrm{s}^-1$. As a next step, the hypergraph $\mathcal{G}$ is generated. A tiling with 25 tiles per component is applied, for a total of $25^4 = 390625$ vertices. The action subset is restricted to the four different deflections $\delta_e \in \mathcal{A}_{sub} = \{-4°,\ -2°,\ 2°,\ 4°\}$. The time-step Δt is chosen as $0.1\mathrm{s}$. Candidate commands $\boldsymbol{a}$ are limited to a length between $k_{min} = 3$ and $k_{max} = 5$ time-steps. The task fails if the constraints on h or α are violated, or otherwise if $\theta \notin [-\pi/4,\ \pi/4]$ or $q \notin [-\pi/2,\ \pi/2]\mathrm{s}^{-1}$, before the 600^{th} iteration. Each episode is initialized from starting conditions:

$$h_0 = 0\ ;\ \theta_0 = 0\ ;\ \alpha_0 = 0\ ;\ q_0 = 0 \tag{5.27}$$

and the actual dynamics of the aircraft are simulated by replacing B_n with a control matrix $B \in \hat{B}$, selected randomly at the start of the episode. Three agents, one equipped with the OM, one with the PM, and one with a random policy, are tested and compared.

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 5.7, a typical behavior for the controller with the OM is showed. Initially, the agent succeeds in keeping the flight path angle γ sufficiently small. However, as altitude decreases, the agent does not compensate for the altitude loss, because in near-time the predicted states are safe. This is

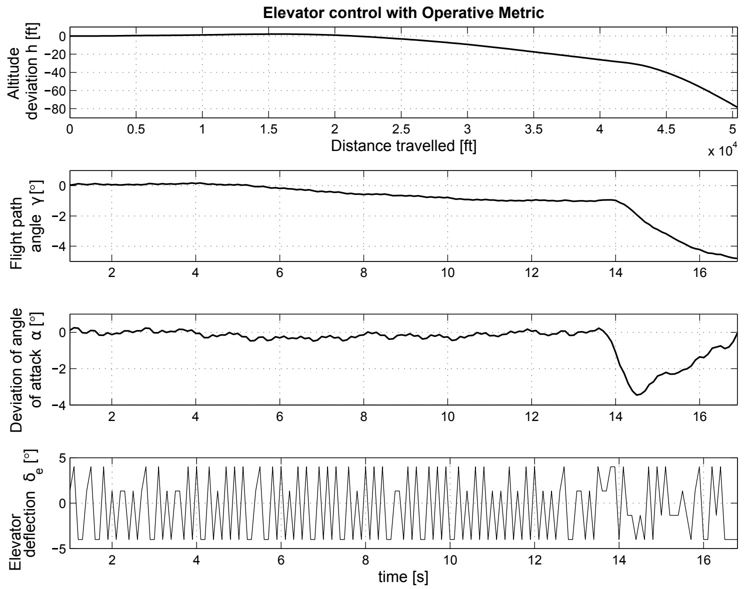


Figure 5.7: A typical behavior for the agent adopting the OM in the elevator control task. The altitude deviation h with respect to distance traveled is shown in the first plot. The second plot shows the change of flight path angle γ with respect to time, while the third plot shows the deviation of angle of attack α with time. The fourth plot shows the elevator deviation.

due to the limited scope of the uncertain predictions. As the system approaches the unsafe boundaries of the altitude range, the commands with the most duration among the candidates (i.e., five time-steps) become unsafe. The agent then adopts commands of shorter duration as feasible candidates. As the boundaries become nearer, the set of feasible commands restricts even more, to commands with a duration of three time-steps. Eventually, all near-time predictions become unsafe (at approximately 13.5 seconds in Figure 5.7). In this event, the metric does not provide any useful information on the commands; the agents then adopts random actions which rapidly lead to a failure of the task.

Proximity metric for elevator control

The gain vector $\boldsymbol{v}_\mathrm{r}$ is selected as

$$\boldsymbol{v}_\mathrm{r} = (v_\mathrm{r}|_h \; v_\mathrm{r}|_\theta \; v_\mathrm{r}|_\alpha \; v_\mathrm{r}|_q) = (6,\ 4,\ 2,\ 1), \tag{5.28}$$

where it can be seen that higher gains are assigned to those components that require more time to be affected by changes in δ_e. A typical trajectory for the agent with the PM is shown in Figure 5.8. The aircraft starts pitching down, and gradually loses 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 PM: the agent starts with no visited states, but as soon as either a positive or negative flight path angle is experienced, the agent tries to keep the system in this flight condition.

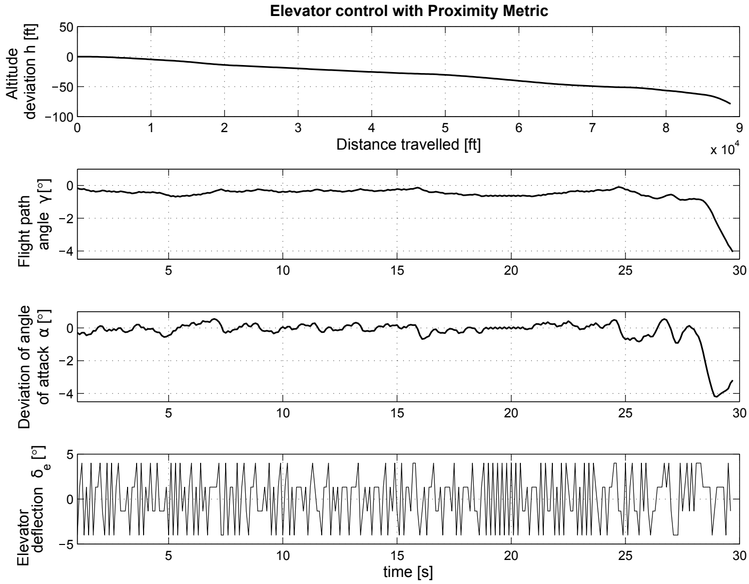


Figure 5.8: A typical behavior for the controller with the PM. The first plot shows the altitude deviation h. The second plot shows the change of flight path angle γ with respect to time, while the third plot shows the deviation of angle of attack α with time. The fourth plot shows the elevator deviation. In this example, the agent does not manage to satisfy the altitude constraint.

Although this effect is mitigated by the higher weight $v_{\mathrm{r}}|_h$ on the altitude, the aircraft does not regain level flight. At around $27\mathrm{s}$, the aircraft reaches the boundaries of the region identified as safe. As with the OM, in Figure 5.8 commands become progressively less safe with the duration. The metric then selects commands with shorter durations, and eventually fails. The agent's policy becomes random, which leads to a violation of the constraints.

Figure 5.9 shows a different behavior with the same agent. During this run the controller manages to keep the flight path angle in $[-1°,\ 1°]$, 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 with the maneuvers of the previous 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 $6\mathrm{s}$. With the OM, the controller achieves 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 manages completion of the task in 22% of the runs, with a minimum duration of 129 iterations, and a mean duration of 350 iterations.

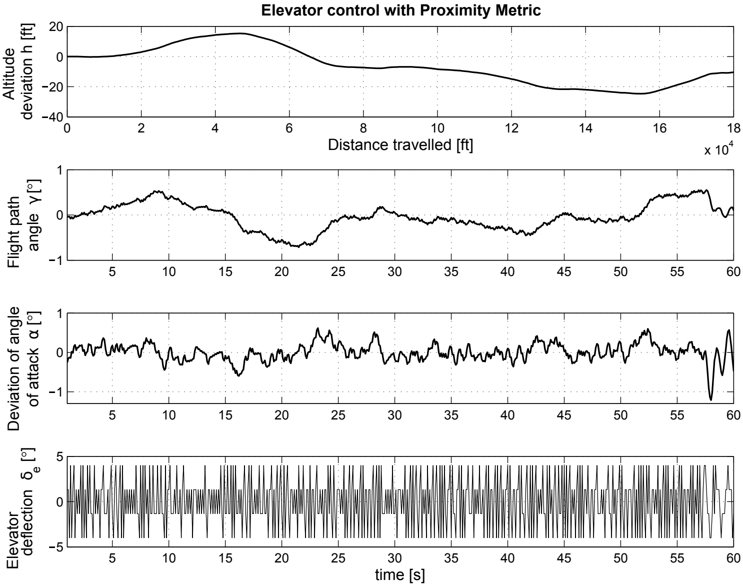


Figure 5.9: A different episode with the application of the PM. The agent manages to maintain a sufficiently reduced flight path angle γ and therefore to achieve safe flight. However, this results in limited exploration of the environment during the task.

5.6. Conclusions and future work

This chapter introduces a new hybrid approach for the exploration of uncertain 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 hypergraph representing the uncertain model of the dynamics of the environment. The 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 method is a safety metric, which evaluates candidate commands of the agent at every time-step. Solving this optimization problem, which is computationally efficient due to graph formulation of the dynamics, selects the action ultimately performed.

Two metrics are proposed: an Operative Metric (OM) assigning values to vertices depending on the current belief of safety, and a Proximity Metric (PM) computing distances between vertices of the graph and previously visited states. Both approaches are 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 OM is found to be effective in achieving safe exploration, avoiding collisions in all runs and exploring the room environment. Conversely, the PM results either in episodes with collisions, or in constant maneuvers with very limited exploration. In the elevator task, the OM is able to enforce safety only for the first instants of flight. The PM performs better by limiting the rate of altitude loss, achieving longer duration of the task.

The results appropriately show the characteristics of the metrics. As expected, the effects of implementing the PM are shown to be analogous to those obtainable by implementing the distance metric with OptiSHERPA: the evolution of the system is restricted, which increases safety but can significantly reduce exploration. The elevator deflection task shows that the PM by itself is not always able to prevent fatal occurrences. In this respect, the addition of a dedicated risk-evasion metric as the one adopted with OptiSHERPA (Section 2.4.2) could be beneficial. Concerning the OM, the results show that the metric is more successful in high-level control tasks, where the prediction horizon is sufficient to account for the problem constraints, such as the holonomic constraints of the quadrotor task. Additionally, in this cases the OM performs better that the PM, which can get stuck in constant maneuvers.

6

Vertex Classification

*This chapter introduces the **Vertex Classification** hybrid method, which simultaneously addresses the challenges of **safety**, **online efficiency** and **robustness**. The method consists of assigning **levels** and **coefficients** to vertices, which are then used to assess the safety of actions. The method combines the assumptions in terms of bounding model and risk perception of Chapter 2, the graph formulation of Chapter 3 and the state projection of Chapter 4. The **safest policy**, implemented in a simulated MAV task, is found to share similarities with potential-based methods, and also entirely prevents any fatal occurrence.*

6.1. Introduction

Chapter 3 introduced the use of graphs in RL for the purpose of increasing the online efficiency of safe exploration algorithms. Additionally, Chapter 3 introduced graph pruning, a procedure to check whether or not a given policy is guaranteed to be feasible, and thus safe, during the entire learning.

However, the more the environment increases in size, dimensionality and uncertainty, the more these methods are difficult to apply. In this chapter, the hierarchical projection of Chapter 4 is applied in order to simplify the graph formulation. Furthermore, *Vertex Classification* is introduced, a new algorithm for safety assessment, which is less conservative and less sensitive to the discretization parameters than the pruning method.

Section 6.2 shows the motivations to apply state projection to operational envelopes. Section 6.3 introduces the Vertex Classification method. Section 6.4 shows how to apply Vertex Classification to an MAV task.

6.2. State projection for graph methods

This section presents the motivation for applying state projection, as introduced in Section 4.3, to graph methods with operational envelopes (OE). A practical example is provided to clarify the state projection procedure. Finally, this section summarizes the necessary assumptions for the remainder of the chapter.

6.2.1. Motivation for state projection

Given state $\boldsymbol{x} = (x_1, \cdots, x_n) \in \mathcal{S}$ and action $u \in \mathcal{A}$, the "original" problem considered in this section is to check if a policy $\pi : \mathcal{S} \rightarrow \mathcal{A}$ is feasible, i.e., if given a known OE $\mathcal{S}_{env} \subset \mathcal{S}$, and the unknown but bounded dynamics $\mathcal{D} \in \hat{\mathcal{D}}$ of the environment, it is $\boldsymbol{x}(t) \in \mathcal{S}_{env}, \forall t$. Chapter 3 addresses this problem through the use of a pruned graph $\mathcal{G}_p$.

However, this can be difficult to apply. Consider for example an OE that is either high dimensional, extensive in size, or both. In this case, the graph $\mathcal{G}$ overestimates $\mathcal{D}$ considerably, since the total number of tiles $\tau \in \mathcal{T}$ is limited by the agent memory and power, and since less refined tilings $\mathcal{T}$ result in less precise estimations. A second inconvenience arises if the environment is partially unknown, which complicates the formulation of an OE, and thus the adoption of pruning or of any similar graph-based method. If the current state $\boldsymbol{x}$ belongs to a known set $\mathcal{S}_{knw}$, the method can be used by adopting $\mathcal{S}_{knw}$ as a temporary OE, and trying to expand that set during exploration; however, each modification of the OE would require another computation of the hypergraph $\mathcal{G}$, as illustrated in Section 3.3.3. Furthermore, if the agent does not succeed in expanding the OE, it might linger within its restricted OE indefinitely, which is safe but also inefficient.

An answer to these concerns was presented in Chapter 4 by means of hierarchy and of state projection. Consider the bijective projection function

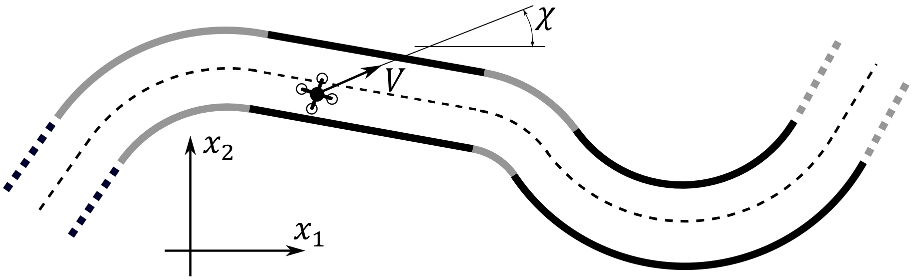


Figure 6.1: The corridor is divided into segments with constant curvature. The original state is composed of absolute position x_1 and x_2, of heading χ, and of speed V.

$$\mu : \mathcal{S} \rightarrow \sum_{i\in\{1,\ldots,m\}}^{\times} \mathcal{P}^i, \tag{6.1}$$

where $\mathcal{P}^i$ are projected state spaces, and

$$\mu_i(\boldsymbol{x}) = \boldsymbol{p}^i \in \mathcal{P}^i, \tag{6.2}$$

are the projected state vectors with projected dynamics

$$\dot{\boldsymbol{p}}^i = \mathcal{D}^i(\boldsymbol{p}^1, \ldots, \boldsymbol{p}^m, u). \tag{6.3}$$

It follows that $\mathcal{S}_{env}$ can also be projected as $\mu(\mathcal{S}_{env}) = \{\boldsymbol{p}^1, \ldots, \boldsymbol{p}^m | \mu^{-1}(\boldsymbol{p}^1, \ldots, \boldsymbol{p}^m) \in \mathcal{S}_{env}\}$. Define then the projected envelope $\mathcal{P}^i_{env}$ as the set

$$\mathcal{P}^i_{env}(\boldsymbol{p}^{j\neq i}) = \{\boldsymbol{p}^i | \mu^{-1}(\boldsymbol{p}^1, \ldots, \boldsymbol{p}^m) \in \mathcal{S}_{env}\}, \tag{6.4}$$

where $\boldsymbol{p}^{j\neq i}$ indicates all projected states other than $\boldsymbol{p}^i$. Therefore, the original problem of guaranteeing $\boldsymbol{x} \in \mathcal{S}_{env}$ is equivalent to the one expressed by

$$\boldsymbol{p}^i \in \mathcal{P}^i_{env}. \tag{6.5}$$

It is assumed here that the elements of the projected space $\mathcal{P}^i$ have components that are relative to or independent of the remaining projected states. Thus, according to Definition 11 and Definition 12, it is

$$\dot{\boldsymbol{p}}^i = \mathcal{D}^i(\boldsymbol{p}^i, \boldsymbol{p}^{i\neq j}|_0, u), \tag{6.6}$$

indicating with $\boldsymbol{p}^{i\neq j}|_0$ an arbitrary set of initial values for the states $\boldsymbol{p}^{j\neq i}$, with respect to which the components of $\boldsymbol{p}^i$ are relative to or independent of. With this assumption, it is possible to compute $\dot{\boldsymbol{p}}^i$ in order to satisfy Eq. (6.5). Nonetheless, it should be noted that, unless $\boldsymbol{p}^i$ is relative to all $\boldsymbol{p}^{j\neq i}$, $\mathcal{P}^i_{env}$ might change with time.

6.2.2. An example of state projection for UAV tasks

Consider now the task depicted in Figure 6.1, similar to the one of Section 3.3.3, and consisting of flying through a corridor with piecewise constant curvatures. The state is $\boldsymbol{x} = (x_1, x_2, \chi, V)$, with x_1 and x_2 the absolute position of the UAV, χ the heading, and V the speed. The control vector is $(u_1, u_2) = (\dot{V}, \dot{\chi})$. The envelope for this task can be informally expressed as those (x_1, x_2, ψ) for which the UAV is not in collision and flies towards the end of the corridor. Solving this problem with a graph-approach can be quite challenging. First of all, it can be difficult to find a partition of x_1, x_2 that correctly discretizes the corridor, depending on its length and shape, without excessively increasing the refinement of the tiling, and thus its memory and computation requirement. Moreover, if the agent does not know the exact geometry of the corridor, the envelope must approximate the local envelope and try to subsequently expand it with exploration.

Consider now the generic segment of the corridor, with constant curvature, and the projection function $\mu(\boldsymbol{x}) : \mathcal{S} \rightarrow \mathcal{P}^1 \times \mathcal{P}^2$ with projected states

$$\boldsymbol{p}^1 = (x_1|_\mathrm{m}, x_2|_\mathrm{m}, \chi|_\mathrm{m}, r_\mathrm{m}) \in \mathcal{P}^1; \; \boldsymbol{p}^2 = (\rho, \theta, \psi, V) \in \mathcal{P}^2. \tag{6.7}$$

The first state $\boldsymbol{p}^1$ identifies the geometry of the segment, and consists of the position $(x_1|_\mathrm{m}, x_2|_\mathrm{m})$ of the midline of the corridor (the dashed line in Figure 6.2) at the start of the segment, of the direction of the midline $\chi|_\mathrm{m}$ in that same position, and of the curvature r_m. The components of the second projected state $\boldsymbol{p}^2$ are instead the cylindrical coordinates ρ and θ with respect to the center of curvature (c_1, c_2), the heading ψ of the UAV with respect to the tangent to the midline, and the speed V. Notice that the center (c_1, c_2) is uniquely defined once the components of $\boldsymbol{p}^1$ are given. The components ρ, θ, ψ, and V of $\boldsymbol{p}^2$ are *relative* to those of $\boldsymbol{p}^\mathbf{1}$, since

$$\mathcal{D}^1 = 0, \tag{6.8}$$

while the last component V of $\boldsymbol{p}^2$ is also *independent* of $\boldsymbol{p}^1$:

$$\frac{\partial \dot{V}}{\partial \boldsymbol{p}^\mathbf{1}} = 0. \tag{6.9}$$

Adopting the projection μ for each segment of the corridor, the envelope can now be defined as

$$\mathcal{P}^2_{env}(\boldsymbol{p}^1, w) = \{\boldsymbol{p}^2 | \; \|\rho - r_\mathrm{m}\| < \frac{w}{2}, \|\psi\| < \frac{\pi}{2}\}, \tag{6.10}$$

where w is the width of the corridor. Due to Eq. (6.8) and Eq. (6.9), given the current value of $\boldsymbol{p}^1$, and by estimating the width w, it is possible to sequentially solve the problem in the sole projected state $\boldsymbol{p}^2$. Assuming lower and upper bounds $[\underline{r}_\mathrm{m}, \overline{r}_\mathrm{m}]$ for the curvature and $[\underline{w}, \overline{w}]$ for the width, adopting a reasonable tiling $\mathcal{T}$ for the projected state $\boldsymbol{p}^2$ is simpler than doing the same for the original state $\boldsymbol{x}$, also due to the difference in size between the two spaces $\mathcal{S}$ and $\mathcal{P}^2$. This greatly simplifies the application of graph based methods. As an extreme case, the problem could be a-priori solved for a number of envelope *beliefs* $\mathcal{P}^2_{env}(\hat{r}_\mathrm{m}, \hat{w})$, where $\hat{r}_\mathrm{m} \in [\underline{r}_\mathrm{m}, \overline{r}_\mathrm{m}]$

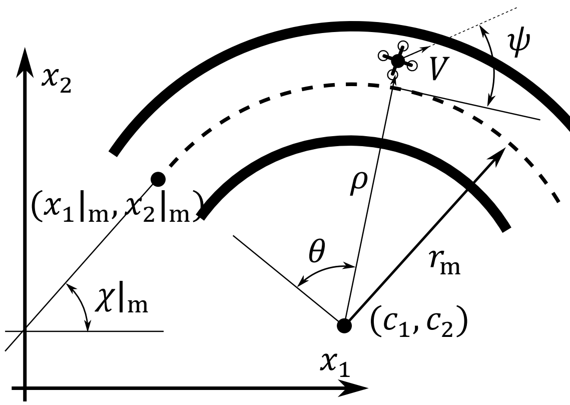


Figure 6.2: The original state $(x_1, x_2, \chi, V) \in \mathcal{S}$ is decomposed as the two projected states $\boldsymbol{p}^1 = (x_1|_\mathrm{m}, x_2|_\mathrm{m}, \chi|_\mathrm{m}, r_\mathrm{m})$ and $\boldsymbol{p}^2 = (\rho, \theta, \psi, V)$.

and $\hat{w} \in [\underline{w}, \overline{w}]$ are estimates of the actual curvature and width of the different segments of the corridor.

6.2.3. Assumptions

Section 6.2.2 shows how applying graph based methods is simpler if there exists a state projection with a state $\boldsymbol{p}^i$ which is relative or independent of the others. It is assumed that the projected envelope $\mathcal{P}_{env} = \mathcal{P}^i$ of the projected state $\boldsymbol{p}^i = (p^i_1, \dots, p^i_n)$ is in the form

$$\mathcal{P}_{env} = [\underline{p}^i_1, \overline{p}^i_1] \times \cdots \times [\underline{p}^i_n, \overline{p}^i_n], \tag{6.11}$$

and that a bounding model of dynamics $\mathcal{D}^i$ exists, so that

$$\hat{\mathcal{D}}^i : \begin{cases} \dot{p}^i_1 \in \hat{\mathcal{D}}^i_1\,(p^i_1, \dots,\ p^i_n,\ u) \\ \qquad\vdots \\ \dot{p}^i_n \in \hat{\mathcal{D}}^i_n\,(p^i_1, \dots,\ p^i_n,\ u), \end{cases} \tag{6.12}$$

similarly to what assumed in Section 3.2.3. $\mathcal{P}_{env}$ is partitioned into a tiling $\mathcal{T}$ with tiles τ (Figure 6.3). A subset of action $\mathcal{A}_{sub}$ and a time interval Δt are selected as well, in order to generate a graph $\mathcal{G}$, in accordance with the procedures of Section 3.2.4 and Section 3.2.5 (Figure 6.4). In the event that $\boldsymbol{p}^i(t + \Delta t) = \boldsymbol{p}^i(t) + \Delta\boldsymbol{p}^i$ is not contained in $\mathcal{P}_{env}$, the edge is not added to the graph. Instead, these combinations (τ, u) are stored in a list of *critical* transitions. Table 6.1 illustrates the differences between the following state representations: the "original" state $\mathcal{S}$, the projected state $\mathcal{P}^i$, and the graph $\mathcal{G}$.

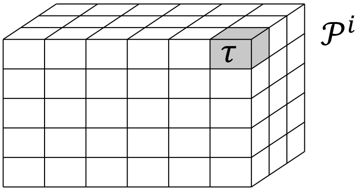


Figure 6.3: A sample tiling $\mathcal{T}$ partitioning of a 3-dimensional projected state space $\mathcal{P}^i$. One tile τ is shaded.

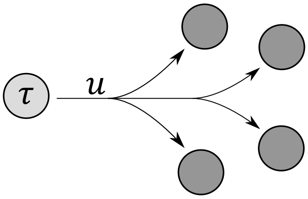


Figure 6.4: An example of the uncertain transitions represented by the edges of $\mathcal{G}$.

	Original	**Projected**	**Graph-Tiling**
State	$\boldsymbol{x} \in \mathcal{S}$	$\boldsymbol{p}^i \in \mathcal{P}^i$	$\tau \in \mathcal{T}$
Action	$u \in \mathcal{A}$	$u \in \mathcal{A}$	$u \in \mathcal{A}_{sub}$
Continuity	yes	yes	no
Projection	no	yes	yes

Table 6.1: Characteristics and properties of the three different state representations.

6.3. The Vertex Classification method

This section explains how to use the graph formulation to perform *Vertex Classification* (VC), which is a method to quantitatively estimate the safety of the states of $\mathcal{P}_{env}$. This estimate can then be used to evaluate a given policy in combination with the graph $\mathcal{G}$.

6.3.1. Desirability of transitions

The graph $\mathcal{G}$ provides an overestimation of all transitions of $\boldsymbol{p}^i$ within $\mathcal{P}^i$, with the exception of critical transitions. Depending on which of the constraints of Eq. (6.11) is violated, it is possible to distinguish between desired and undesired critical transitions. Consider the corridor example of Section 6.2.2 with $\boldsymbol{p}^i = (\rho, \theta, \psi, V)$ and envelope

$$\mathcal{P}_{env} = [r_{\mathrm{m}} - \frac{w}{2}, r_{\mathrm{m}} + \frac{w}{2}] \times [0, \theta_{\max}] \times [-\frac{\pi}{2}, \frac{\pi}{2}] \times [\underline{V}, \overline{V}], \tag{6.13}$$

where $\theta_{\max}$ is the value of θ at the end of the segment of corridor and $[\underline{V}, \overline{V}]$, $\underline{V} \geq 0$ is an arbitrary constraint on V in order to comply with Eq. (6.11). Violating the constraint in ρ must be avoided as it results in a collision: all such edges are added to a list U of *undesired* critical transitions. Conversely, violating only the constraint $\theta < \theta_{\max}$ means that the UAV has managed to traverse the current sector into the new one, coming closer to the end of the corridor. Therefore, these *desired* critical transitions are gathered in a list D. The remaining critical transitions are assumed to be neither desired or undesired and are therefore ascribed to a neutral set N.

For the general case, the designer must provide the agent with a *desirability* criteria, so that critical transitions can be grouped into sets with equal desirability. For the remainder of the chapter, it will be assumed that at least the two aforementioned sets U and N, and possibly one or more sets D_i, are identified. This specification is a primary concern of the method, as the goal of VC is promoting desired transitions and preventing undesired ones.

6.3.2. Vertex level and coefficient assignments

Vertex Classification entails assigning to each vertex two values: its *level* and its *coefficient*, which will quantify the safety of the vertex itself. This method is shown in Algorithm 6, where l and c indicate levels and coefficients of vertices. Consider the set $U \in \mathcal{T} \times \mathcal{A}_{sub}$, and assign level 0 to each vertex $\tau : \forall u, (\tau, u) \in U$. These are the most unsafe vertices. Assign then level 1 to each vertex not already of level 0 that is connected by any edge to a level 0 vertex. These are vertices that could provoke an undesired transition in at least two time intervals. The procedure is repeated, increasing level by one at each iteration, until all vertices have been assigned a level. This will be a first indication of the safety of the vertex itself, representing the minimal amount of time-steps that can be guaranteed before a critical transition occurs. To save computation time, it is possible to terminate the labeling at any iteration by assigning a maximum level $l_{\max}$ to all remaining vertices.

Coefficients are then assigned according to levels. As a start, assign a coefficient of 1 to each level 0 vertex. Then, coefficients for level 1 are assigned as follows. Select a level 1 vertex and consider its connected vertices. Sum the coefficients of all level 0 vertices among these, then divide it by the total number of connected vertices. For level 0 vertices, this is equivalent to assigning the ratio between the number of level 0 connected vertices and the total number of connected vertices. It follows that for two vertices of equivalent level, the one with lower coefficient is on average the furthest from an undesired transition. Assigning of coefficients for vertices of higher level is performed in the same way, summing the coefficients of vertices with the lowest level, and dividing it by the total number of connected vertices. Figure 6.5 shows an example of level and coefficient computation.

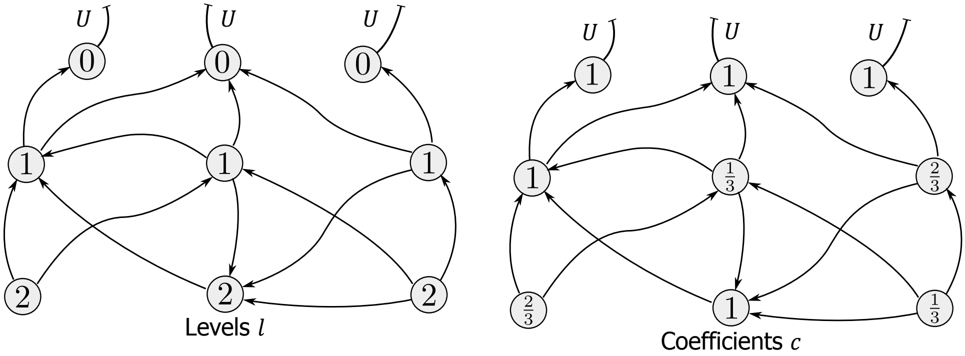
In the event that one or more sets of desired transitions, indicated by D_i, are considered, the same procedure can be applied for each set to identify one or more additional series of levels and coefficients. In this case, the lower the level and the higher the coefficient, the nearer to desired transitions. Finally, neutral transitions are not considered for the assignment of coefficients and levels.

6.3.3. Action selection

After classification, each vertex is labeled with a level and a coefficient that describe its safety. This information can be used to assess the safety of an action, and thus of a policy. In the event that two sets U and N are given, the safety $s(\tau, u)$ of edge (τ, u) is given by

$$\mathrm{s}(\tau, u) = -\frac{\sum_{\tau' \in \mathcal{G}(\tau,u)} c(\tau') \cdot \omega^{l^U_{\max} - l(\tau')}}{|\mathcal{G}(\tau, u)|}, \tag{6.14}$$

where $l^U_{\max}$ indicates the maximum level among all vertices, and where $|\mathcal{G}(\tau, u)|$ indicates the number of vertices τ' connected to τ through edge (τ, u). The parameter $\omega > 1$ is used to modulate the conservativeness of actions: the greater the



Levels l

Coefficients c

Figure 6.5: An example of computation of levels and coefficients. Levels are assigned progressively, starting from vertices for which all transitions are undesired (top). Coefficients are assigned depending on levels of connected vertices, and the higher the coefficient, the nearer undesired transitions. For example, among the $l = 2$ vertices, it can be seen how the vertex with $c = \frac{1}{3}$ is safer than the one with $c = \frac{2}{3}$ and definitely safer than the one with $c = 1$.

Algorithm 6 Vertex Classification

```
Initialize
    $\mathcal{G}$, $U$, $\mathcal{T}$, $\mathcal{A}_{sub}$, max_iterations
    lbl ← 0
    set ← $\{\tau | \forall u, (\tau, u) \in U\}$
    $l$(set) ← 0
    labeled ← set
while lbl ≤ max_iterations ∧ labeled ≠ $\mathcal{T}$ do
    set ← $\mathcal{G}^{-1}$(set, $\mathcal{A}_{sub}$) \ labeled
    lbl ← lbl + 1
    $l$(set) ← lbl
    labeled ← labeled ∪ set
set ← $\{\tau | l(\tau) = 0\}$
$c$(set) ← 1
lbl ← 1
for $j = \{1, \ldots, \max(l(\mathcal{T}))\}$ do
    prv_set ← set
    set ← $\{\tau | l(\tau) = j\}$
    for $\tau \in$ set do
        $\mathcal{G}_{\max}$ ← $\mathcal{G}(\tau, \mathcal{A}_{sub})$ ∩ prv_set
        $c(\tau)$ ← $\frac{\sum c(\mathcal{G}_{\max})}{|\mathcal{G}(\tau, \mathcal{A}_{sub})|}$
```

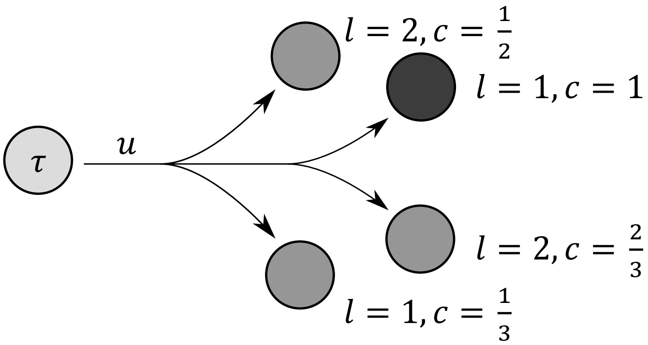


Figure 6.6: An example of safety computation for an edge (τ, u) with four vertices. Given potential $\omega = 3$ and maximum level $l_{\max}^{U} = 4$, the safety of action u in τ is equal to $-\frac{93}{8}$.

value, the more the risk of decreasing a level is penalized, and therefore the lower the safety of that action. If one or more sets of desired transitions D_i are present as well, it is

$$\mathrm{s}(\tau, u) = \frac{\sum_{\tau' \in \mathcal{G}(\tau,u)} \left(\sum_i c_{D_i}(\tau') \cdot \omega_{D_i}^{l_{\max}^{D_i} - l_{D_i}(\tau')} \right) - c_U(\tau') \cdot \omega_U^{l_{\max}^{U} - l_U(\tau')}}{|\mathcal{G}(\tau, u)|}. \tag{6.15}$$

Terms $l_{\max}^{D_i}$ indicate the maximum level of vertices with respect to sets D_i. Notice that each set U, D_i has its own parameter ω. Figure 6.6 provides an example of safety computation as in Eq. (6.14). Edge (τ, u) connects to four vertices, with levels l equal to $2, 1, 2, 1$ and coefficients c equal to $\frac{1}{2}, 1, \frac{2}{3}$ and $\frac{1}{3}$. Assuming $\omega = 3$ and a maximum level $l_{\max}^{U} = 4$, it is then

$$s(\tau, u) = -\frac{1}{4} \cdot (\frac{1}{2} \cdot 3^{4-2} + 1 \cdot 3^{4-1} + \frac{2}{3} \cdot 3^{4-2} + \frac{1}{3} \cdot 3^{4-1}) = -\frac{93}{8}. \tag{6.16}$$

Looking at the different contributions, it can be seen that the transition to the black vertex in Figure 6.6, with $l = 1$ and $c = 1$, is decisive for $s(\tau, u)$, accounting for 58% of the calculated value. This is in accordance with the definition of safety, since from that vertex a critical transition would certainly follow, at the next time-step (see Figure 6.5).

Computing safety $s(\tau, u)$ as in Eq. (6.16) can be used to evaluate the safety of policies as well. For example, the agent can compare two policies based on the average or on the worst-case safety of their edges. Otherwise, s could be used to negate actions with safety inferior to an arbitrary threshold, or as an additional weighting term for action-state values in more conventional algorithms, such as Q-learning [6].

Comparing VC with the pruning approach of Chapter 3, several differences emerge. First, pruning is more preventive with respect to undesired transitions, as it directly removes edges (actions) from the graph. As a result, however, pruning does not guarantee to find feasible policies; also, if feasible policies are found, these can be very conservative and restrictive towards the agent. Another improvement of VC is that, when pruning is adopted, much attention must be taken in selecting the refinements Δ_i and the timestep Δt used to discretize the dynamics, since these parameters influence how the graph representation artificially accelerates the dynamics of the system, as illustrated in Section 3.2.4. Since pruning is a conservative process, the more the dynamics are accelerated, the more a graph will be pruned starting from the same sinks. VC is less sensitive to this effect, since actions are not directly removed from the action set.

The safety estimates $s(\tau, u)$ can be used to define a *safest policy*

$$\pi^{\mathrm{s}}(\tau) = \mathrm{argmax}_{u\in\mathcal{A}_{sub}}\mathrm{s}(\tau, u), \tag{6.17}$$

which selects actions in order to minimize the chance of undesired transitions, from which agent is repelled, and increase that of desired ones, which conversely attract the agent. As it will be shown in Section 6.4.6, this effect of the policy on the MAV agent resembles that of a potential field [89]. For this reason, ω are said to be the *intensities* of the policy. The analogy with the field can also be of use to interpret and set these parameters.

6.4. Vertex Classification for the MAV task

This section introduces the application of VC to an MAV task derived from the one of Section 4.5.1, which constitutes a case study for the method. The state projection adopted, the resulting agent implementation, and the interpretation of the levels and of the coefficients are discussed here.

6.4.1. Task description

First, the MAV task is briefly summarized. Given state $\boldsymbol{x} \in \mathcal{S}$ consisting of absolute position x_1, x_2, heading χ and speed V, the available bounding model $\hat{\mathcal{D}}$ is

$$\begin{gathered}\dot{x}_1 = \hat{\eta}V\cdot\cos(\chi)\ ;\ \dot{x}_2 = \hat{\eta}V\cdot\sin(\chi)\ ;\ \dot{\chi} = \hat{q}\cdot u_1\ ;\\ \dot{V} = \begin{cases}\max(0, \hat{a}\cdot u_2) & \text{if } V = \underline{V}\\ \hat{a}\cdot u_2 & \text{if } V \in (\underline{V}\ , \overline{V})\ .\\ \min(0, \hat{a}\cdot u_2) & \text{if } V = \overline{V}\end{cases}\end{gathered} \tag{6.18}$$

with $\underline{V} = 0.1\frac{\mathrm{m}}{\mathrm{s}}$ and $\overline{V} = 0.3\frac{\mathrm{m}}{\mathrm{s}}$, $(u_1, u_2) \in [-1\ ,\ 1]$, $\hat{q} = [\underline{q}, \overline{q}] = [17, 30]\frac{\circ}{\mathrm{s}}$ and $\hat{a} = [\underline{a}, \overline{a}] = [0.1, 0.3]\frac{\mathrm{m}}{\mathrm{s}^2}$, $\hat{\eta} = [\underline{\eta}, \overline{\eta}] = [0.9, 1.1]$.

The task is performed in the walled room environment, already presented in Section 4.5.1 (see Figure 6.7 and Figure 6.8). The MAV sensors detect obstacles within $1.5\mathrm{m}$ at an angle between -100° and 100° from the MAV facing. Contrary to the same task in Section 4.5.1, here the MAV does not have to learn the position of the goal; the direction to the goal χ_{goal} is assumed to be known to the agent at all times. The task of the MAV is

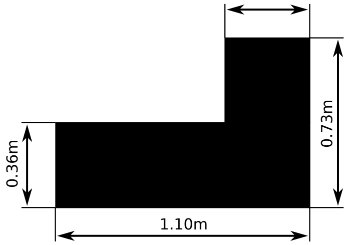


Figure 6.7: Shape of elementary obstacles.

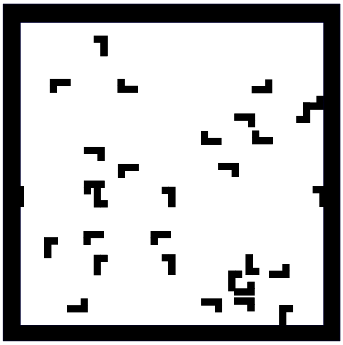


Figure 6.8: Example of a room environment for the task.

1. to avoid collisions (safety constraint);
2. in the absence of obstacle, to fly towards the goal position (performance constraint).

The VC method is applied to obtain a policy which satisfies the safety constraint. In absence of impeding obstacles, the requirement of moving towards the goal is satisfied by opportunely selecting the projected envelope, as explained in Section 6.4.7.

6.4.2. Envelope definition and state projection

The OE corresponding to the task is formulated as

$$\mathcal{S}_{env} = \{\boldsymbol{x}|\ x_1, x_2 \in [-8\ ,\ 8],\ V \in [\underline{V}\ ,\ \overline{V}], (x_1, x_2) \notin \texttt{obs}\}, \tag{6.19}$$

where obs represents impassable locations such as walls and obstacles. The condition on x_1, x_2 represents the safety constraint.

The difficulty of partitioning the envelope derives once again from the limited knowledge of the environment, in this case because obs is unknown. However, the state projection of Section 4.3 can simplify the above task. The projection $\mu(\mathcal{S}) = \mathcal{P}^1 \times \mathcal{P}^2$, with

$$\boldsymbol{p}^1 = (x_1|\mathrm{c}, x_2|\mathrm{c}, \chi|_\mathrm{c}) \in \mathcal{P}^1\ ;\ \boldsymbol{p}^2 = (\rho, \theta, \psi, V) \in \mathcal{P}^2, \tag{6.20}$$

is a coordinate change from the absolute state (x_1, x_2, χ) to a relative state determined by $\mathcal{P}^2$ with respect to a reference given by $\mathcal{P}^1$:

$$\begin{aligned} \rho =& \|\ (x_1 - x_1|_\mathrm{c}, x_2 - x_2|_\mathrm{c})\ \|; \\ \theta =& \arctan\left(\frac{x_2 - x_2|_\mathrm{c}}{x_1 - x_1|_\mathrm{c}}\right) - \chi|_\mathrm{c}; \\ \psi =& \chi - \chi|_\mathrm{c} - \theta, \end{aligned} \tag{6.21}$$

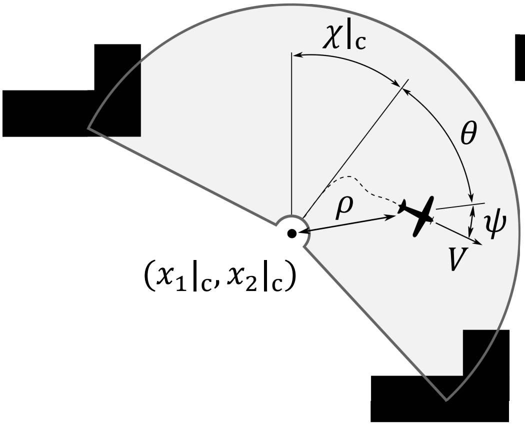


Figure 6.9: The new state description $\mu(\mathcal{S}) = \mathcal{P}^1 \times \mathcal{P}^2$, with $\boldsymbol{p}^1 = (x_1|c, x_2|_c, \chi_c)$ and $\boldsymbol{p}^2 = (\theta, \rho, \psi, V)$.

already applied in Section 4.14. Figure 6.9 shows this state projection: components $\theta \in [-100°\,,\,100°]$ and $\rho \in [0.1\text{m}\,,\,1.5\text{m}]$ give the relative position in cylindrical coordinates. As before, relative orientation $\psi \in [-\pi, \pi]$ defines the heading together with θ. $V \in [\underline{V}, \overline{V}]$ is again the absolute speed. As it can be seen, the components of $\boldsymbol{p}^2$ are either relative to or independent of $\boldsymbol{p}^1$. The projected OE, $\mathcal{P}_{env}$, is

$$\mathcal{P}_{env} = \mathcal{P}^2 = [0.1\text{m}\,,\,1.5\text{m}] \times [-100°\,,\,100°] \times [-\frac{\pi}{2}, \frac{\pi}{2}] \times [\underline{V}, \overline{V}]. \tag{6.22}$$

It must be noted that the constraint $\psi \in [-\frac{\pi}{2}, \frac{\pi}{2}]$ is introduced arbitrarily to force the MAV to eventually leave the envelope, possibly towards the goal. This constraint is reproduced during flight by imposing $\psi = \pm\frac{\pi}{2} \rightarrow u_1 \neq \pm 1$.

This particular projection is used to solve the original problem iteratively, as in the previous corridor example. The underlying strategy consists in observing the obstacles in proximity of the MAV and avoiding them while still flying towards the goal; this is repeated until the goal is reached. This choice is motivated by the presence of sensors which provide information on nearby obstacles.

6.4.3. Graph generation for the projected state

In order to apply VC, the envelope and the graph $\mathcal{G}$ are necessary. As done in Section 4.5.3, the projected envelope is split into a *left side* $\theta \in [-100°\,,\,20°]$ and a *right side* $\theta \in [-20°\,,\,100°]$, due to the symmetry of the envelope and of the dynamics.

The graph is then generated for the right side, i.e., $\theta \in [-20°\,,\,100°]$ of the envelope. The obtained graph is valid in the left side as well, given that the correct symmetry between tiles is applied. The following tiling, with 12 intervals for $\theta \in [-20°\,,\,100°]$, 12 for $\rho \in [0.1\text{m}\,,\,1.5\text{m}]$, 7 for $\psi \in [-\frac{\pi}{2}\,,\,\frac{\pi}{2}]$, and 4 for V, is applied:

$$\begin{aligned}\mathcal{T} = \{&[-20^\circ\,,\,-10^\circ]\,,\,\cdots\,,\,[90^\circ\,,\,100^\circ]\}\times\\ &\times\{[0.1\mathrm{m}\,,\,0.1\mathrm{m}+\frac{1.4\mathrm{m}}{12}]\,,\,\cdots\,,\,[1.5\mathrm{m}-\frac{1.4\mathrm{m}}{12}\,,\,1.5\mathrm{m}]\}\times\\ &\times\{[-\frac{\pi}{2}\,,\,-\frac{\pi}{2}+\frac{\pi}{7}]\,,\,\cdots\,,\,[\frac{\pi}{2}-\frac{\pi}{7}\,,\,\frac{\pi}{2}]\}\times\\ &\times\{[0.1\frac{\mathrm{m}}{\mathrm{s}}\,,\,0.15\frac{\mathrm{m}}{\mathrm{s}}]\,,\,\cdots\,,\,[0.25\frac{\mathrm{m}}{\mathrm{s}}\,,\,0.3\frac{\mathrm{m}}{\mathrm{s}}]\}.\end{aligned}\tag{6.23}$$

Proceeding with the discretization, control action is reduced to $\mathcal{A}_{sub} = \{(u_1, u_2)|u_1 \in \{-1; 0; 1\}, u_2 \in \{-1; 0; 1\}\}$, and $\Delta t = 0.2s$ is selected.

6.4.4. Obstacle individuation and beliefs

It is assumed that the obstacles can be perceived or at least estimated by the agent through its sensors. With this information, two possible approaches are available. One approach consists of generating the graph and solving the projected envelope problem using the in-flight sensor readings. By detecting or estimating the obstacles, the "occupied" tiles τ_{occ} of $\mathcal{T}$ are detected. Then, those edges (τ, u) that transition to τ_{occ} are assigned to the list of undesired transitions U upon which VC is performed. The refinement of the tiling compatible with this approach depends on how fast the agent can compute the VC weights online. The second approach is to pre-compute the VC weights for a limited amount of envelopes, i.e., of beliefs, and then adopt the weights of the belief that resembles current sensor readings the most.

Both approaches present strengths and drawbacks. Assuming that the sensor information is reliable, the first strategy reduces the number of occupied tiles τ_{occ} to a minimum and is therefore less restrictive and more precise in the weight assignment. However, if the refinement of the tiling itself is low, because of the necessity of computing the weights online, this advantage can be lost. The major drawback of the second approach lies in its beliefs. The more these are representative and comprehensive, the more effective will be the approach. In this chapter, VC is applied according to the second approach.

In order to create the set of beliefs, sensor information is approximated as the five sectors of Section 4.5.3; these are 40° wide each, so that the two sides, left and right, contain respectively the 1^{st}, 2^{nd}, 3^{rd} sectors and the 3^{rd}, 4^{th} and 5^{th} sectors of Figure 6.10. *Blocked* sectors might contain obstacles, *empty* sectors cannot. The sector within which lies the direction to goal is the *goal* sector. This provides a total of $3 \cdot 2^3 = 24$ beliefs, in terms of blocked and goal sectors.

6.4.5. Transition desirability for MAV task

This section individuates the desirability of the different transitions of the envelope. One possible critical transition of $\mathcal{P}_{env}$ is for $\rho > 1.5\mathrm{m}$. If, at the moment of critical transition, θ lies in a blocked sector, this transition is undesired: $(\tau, u) \in U$. Otherwise, if θ lies in an open sector, it is $(\tau, u) \in D_1$. If θ lies in the goal sector, and this is not blocked, (τ, u) is also added to a second list of desired transitions

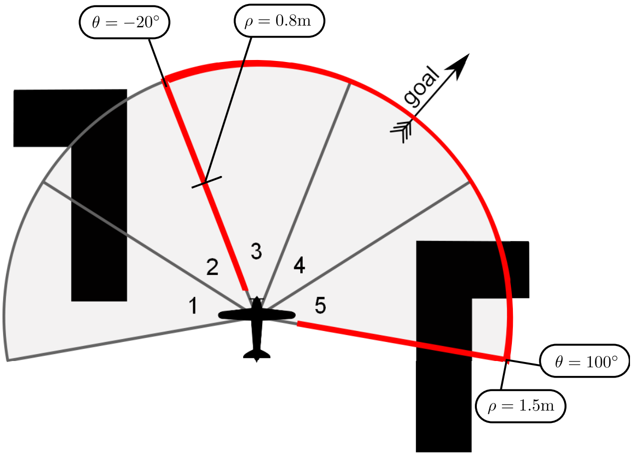


Figure 6.10: The sensed region is split into sectors numbered from 1 to 5. In this example, sectors 1, 2 and 5 are blocked as they contain obstacles. Between the two sides, i.e., sectors (1,2,3) and (3,4,5), the agent selects the right one to navigate, since it contains the least amount of blocked sectors. In addition it also contains the direction to goal χ_{goal}.

D_2. If a critical transition occurs for $\theta < -20°$, and if $\rho \geq 0.8\text{m}$, this edge is also added to U. Note that the MAV is not be considered to violate the envelope if it enters or flies within a blocked sector, even though this might contain an obstacle and therefore lead to a collision. Whether a sector is blocked or not is relevant only if the constraint on ρ is violated. This is apparently in contrast with the definition of critical transition, since collisions can happen inside a blocked sector, and not only at the border. However, a few considerations apply. First, the tiles within a blocked sector are penalized with the suggested implementation as well, since they are more likely to lead to an undesired transition, as it is reflected in their levels and coefficients. Second, consider the alternative solution of indicating every transition to a blocked sector as critical. If VC is applied under this condition, the tiles belonging to a blocked sector would receive level $l = 0$ and coefficient $c = 1$. This would effectively indicate that all vertices of that sector are equally dangerous for the agent. This is not beneficial for assessing which action is the safest, in the event that the agent enters a blocked sector. Instead, with the chosen conditions, the agent receives information from the levels and coefficients of the reachable tiles on how to re-enter a free sector.

6.4.6. Vertex Classification

As a last step, VC is applied, once for each belief, as described in Algorithm 6, setting the maximum number of levels for all sets to nine: $l^{U}_{\max} = l^{D_1}_{\max} = l^{D_2}_{\max} = 9$. Each unoptimized VC for the case study requires between 10 and 15 minutes[1]. Intensities ω_U and ω_{D_1} are empirically selected respectively as 9 and 8.9. Repulsive intensity ω_U is chosen marginally higher than ω_{D_1} to ensure the MAV is repelled

[1] on an Intel Core i5-3360M CPU, 2.80GHz

from blocked sectors even when these are adjacent to attractive, empty sectors, in accordance with the field analogy. Intensity ω_{D_2} is then selected as 5: this results in a moderate attraction on the MAV towards the goal direction.

The safest policy, obtained with these weights according to Eq. (6.17) is tested for the 24 sector combinations, and for 27 dynamics realizations. These are obtained by exhausting the combinations between

$$\left(q \in \left\{\underline{q}, \frac{\underline{q}+\overline{q}}{2}, \overline{q}\right\}, a \in \left\{\underline{a}, \frac{\underline{a}+\overline{a}}{2}, \overline{a}\right\}, \eta \in \left\{\underline{\eta}, \frac{\underline{\eta}+\overline{\eta}}{2}, \overline{\eta}\right\}\right). \tag{6.24}$$

The simulations are initialized with $\theta_0 = 0°$, $\rho_0 = 0.1$, $\psi_0 = 0°$, while the speed is randomly initialized as $V_0 \in [\underline{V}, \overline{V}]$. Results are presented in Figure 6.11, showing four beliefs with less than two blocked sectors, and in Figure 6.12, with two or more blocked sectors. Blocked and empty sectors are indicated as such, and sectors containing or nearest to χ_{goal} are indicated as **TARGET**.

The agent's safest policy π^{s} results predominantly in transitions to target when less than two sectors are blocked, with the exception of Figure 6.11c. In this case, the agent attraction towards the target sector is counteracted by the nearby undesired transition of the blocked sector, as well as by the undesired transition $\theta < -20°$. As a result, only one realization reaches the target, while the others deviate towards an empty set. A similar behavior can be observed in Figure 6.12, where the beliefs with more than one blocked sector are showed. It can be seen that, in these cases, the policy results in undesired transitions as well: nine in Figure 6.12a, and three in Figure 6.12c. Regardless of the specific amount of sectors, the behavior of the agent following π^{s} is as if the sets D_i and U emanate potentials: the agent is attracted to empty sectors, and specifically to the target sector, while at the same time being repelled by the blocked ones.

6.4.7. Implementation of state projection

This section presents the implementation of the safest policy for the MAV task, derived from the implementation of Section 4.5.3. At the start of each episode, the MAV and the goal are positioned at opposite corners of a randomly generated room. Speed and heading of the MAV are initialized randomly: $V_0 \in [0.1\frac{\text{m}}{\text{s}}, 0.3\frac{\text{m}}{\text{s}}]$, $\chi_0 \in [-\pi, \pi]$. Then, a model realization $\mathcal{D}$: $(\eta \in \hat{\eta}, q \in \hat{q}, a \in \hat{a})$ is randomly selected, representing the actual dynamics of the MAV.

The agent controls the MAV according to π^{s}. However, as is the case in Section 4.5.3, this policy is defined only for three sectors. Therefore, before applying π^{s}, the agent chooses the *side of flight*: either the right-side (sectors 3, 4, 5, see Figure 6.10) or the left-side (sectors 1, 2, 3). As in Section 4.5.3, the decisive factor in this application is the amount of blocked sectors: the selected side is the one which contains the least amount of blocked sectors. In the event that the two sides contain the same number of blocked sectors, the chosen side is the one containing the goal direction χ_{goal}.

Once the side of flight is selected, the state is projected according to Eq. (6.20), and Eq. (6.21) is adopted. Indicating as $x_1(k_p)$, $x_2(k_p)$ and $\chi(k_p)$ the absolute

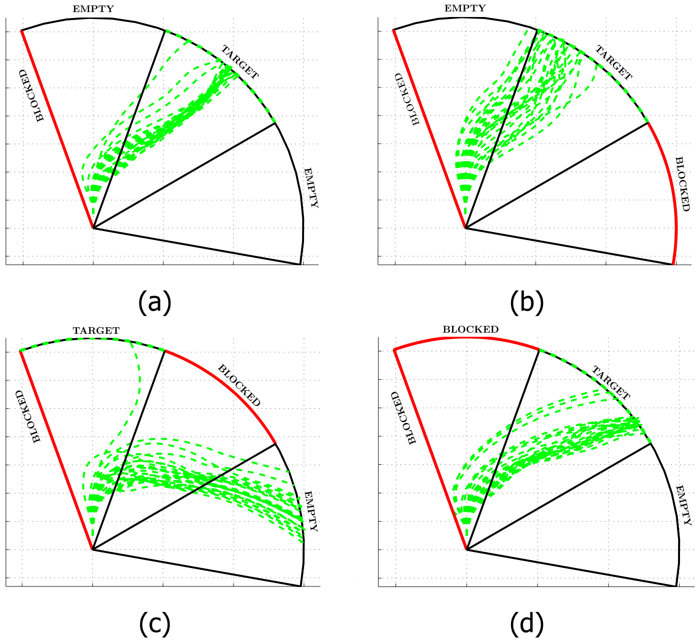


Figure 6.11: The four beliefs with the least amount of blocked sectors. It can be seen that the MAV is able to avoid undesired transitions. The repulsive/attractive nature of the sectors can be observed.

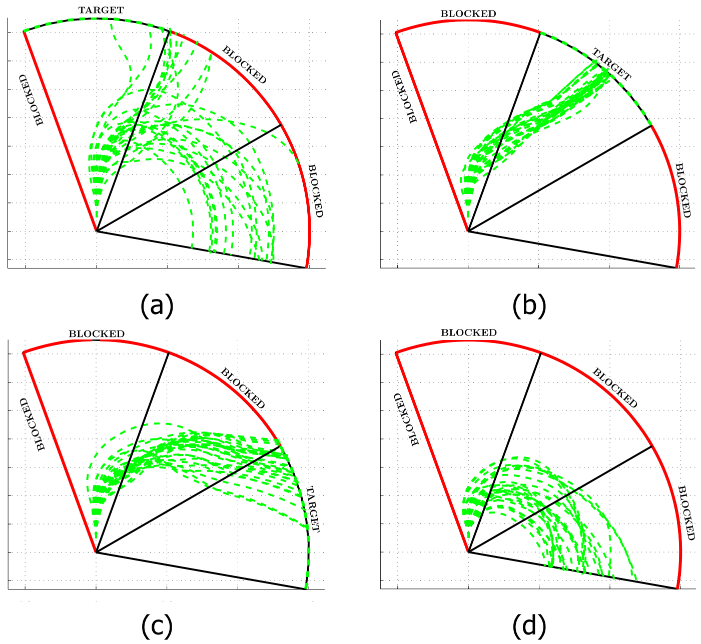
position and heading of the MAV at time of projection k_p, $\boldsymbol{p}^1 \in \mathcal{P}^1$ is chosen as

$$x_1|_{\mathrm{c}} = x_1(k_p) - 0.1\cos\chi\ ;\ x_2|_{\mathrm{c}} = x_2(k_p) - \sin\chi\ ;\ \chi_{\mathrm{c}} = \chi(k_p). \tag{6.25}$$

In this way, every time that the state is projected, the agent is initially in $\boldsymbol{p}^2(k_p) = (0°, 0.1\mathrm{m}, 0°, V)$. This corresponds to the initial condition of the test trajectories of Section 6.4.6.

The sensor information, in terms of blocked and goal sectors, at the time of projection determines the belief of the agent. It then implements the policy obtained through VC for the current belief as its safest policy. In the event that the direction to goal is not contained in the side of flight, the goal sector is selected as either sector 2 or 4, depending on the side. The action $(u_1, u_2) = \pi^{\mathrm{s}}(\tau)$ is then applied, according to current scenario, for the duration of a timestep $\Delta t = 0.2s$, observing the constraint on u_1 so that $\psi \in [-\frac{\pi}{2}, \frac{\pi}{2}]$. In the event that the left side is selected, since π^{s} is defined only for the three sectors of the right side, the projected state is mirrored: $\theta \leftarrow -\theta$, $\psi \leftarrow -\psi$. The corresponding action $\pi^{\mathrm{s}}(\tau)$ is mirrored as well: $u_1 \leftarrow -u_1$.

As detailed in Section 4.5.3, the sensor information that the MAV obtains in flight is used to force a new projection as the observed environment changes, e.g., as new obstacles are detected. At each timestep, the status of the three sectors in the side of flight is compared with that of the previous. If the belief of the agent



(a)

(b)

(c)

(d)

Figure 6.12: The four beliefs with the most amount of blocked sectors. It can be seen that some undesired transitions occur: 8 out of 27 transitions of scenario (a), and 3 out of 27 of scenario (c).

is the same, i.e., if sectors that were previously blocked or goal sectors are still so, the state is not projected, and agent continues to follow its policy π^s. Conversely, if the belief changes, the previous policy is discarded, a new side of flight is selected, the reference $\{x_1|_c, x_2|_c, \chi|_c\}$ is replaced, and a new safest policy is derived. Figure 6.13 shows an example. The state is projected again whenever $\rho \geq \frac{\rho+\overline{\rho}}{2}$, as in Section 4.5.3. This is done to smoothen the MAV trajectory and to facilitate heading changes. The episode ends if the MAV either collides with an obstacle or a wall, or if it reaches within $1\mathrm{m}$ from the goal.

6.4.8. Results with VC safest policy

This section presents the experimental results obtained through simulations of VC applied to the MAV task. Figure 6.14 shows typical results when implementing π^s. It can be seen how the MAV manages to avoid obstacles, and how the attractive/repulsive effect of sectors is conserved when the agent is in the room environment, resulting in a reactive behavior. It can also be noted how the MAV repeatedly performs small turns even in absence of nearby obstacles. This is in accordance with the trajectories of Figure 6.11 and Figure 6.12.

Figure 6.15 shows the presence of *local minima*, a phenomenon typical of path planning with artificial force field (AFF) methods [89]. In AFF methods, local minima are the result of the attractive force of the goal balancing the repulsive forces of

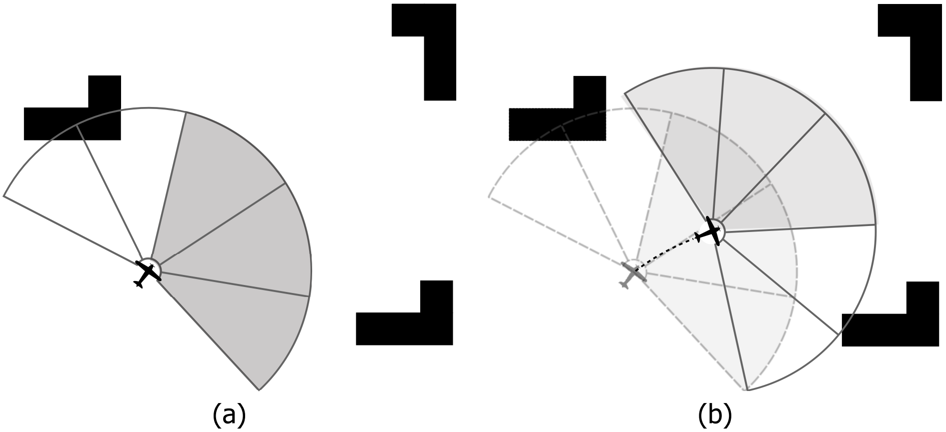


Figure 6.13: An example of the policy update procedure. In (a), the MAV agent selects the three sectors on the right, as no obstacle is perceived. In (b), however, an obstacle in the middle sector, previously free, is detected. Previous policy is then discarded and projection $\boldsymbol{p}^1$, $\boldsymbol{p}^2$ is updated. Now the MAV agent switches to the left side, as it is free of obstacles.

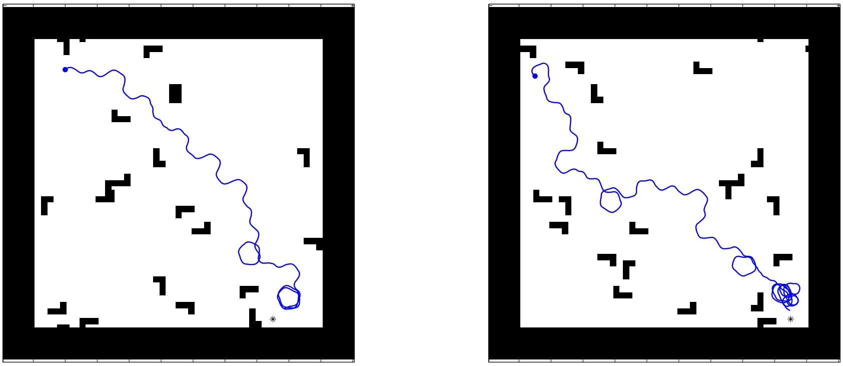


Figure 6.14: Two sample trajectories for the MAV inside the unknown room environment. The dot represents the starting position of the MAV, while the star is the goal. In both cases the MAV reaches the goal.

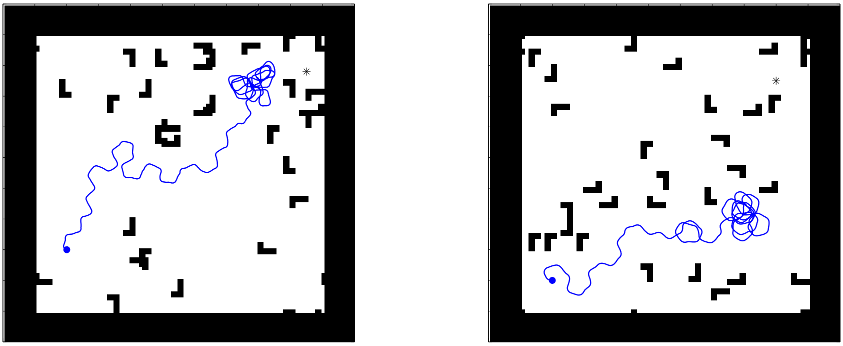


Figure 6.15: In these examples, the MAV does not reach the goal as if caught in a potential local minimum.

the obstacles, therefore interrupting motion or causing oscillations. Here, these occurrences are not due to the presence of opposing forces, but consist of a dynamic process of diversion from the goal given by the obstacles, and successive attraction due to the goal. As a result the MAV continuously turns towards the goal, and then away from it. Solving this problem is beyond the immediate scope of this chapter; the only observation that will be made here is that this kind of occurrences confirms the analogy between VC and AFF approaches.

Lastly, a different setting is simulated in order to confirm the independence of the method from the specific environment, and from the specific realization $\mathcal{D} = (\eta, q, a)$. In this setting, the MAV and the goal are initially positioned in two arbitrary locations of a randomly generated room environment, and control is performed as before. However, every ten state projections, the goal is randomly repositioned, and the current model realization is replaced by a new random tuple (η, q, a). The experiment is repeated for multiple episodes, with a maximum of 500 state projection per episode; after five episodes, a new room environment is generated, and a new episode starts as described above. A total of $1.06 \cdot 10^5$ time-steps are simulated this way, equal to approximately six hours of flight. In spite of the differences in environment, initialization, and model realization, no collisions have been observed. Two sample trajectories obtained during the experiment are shown in Figure 6.16. As it can be seen, the behavior of the MAV is in accordance with the one previously shown.

6.5. Conclusions

This chapter presents how to use the state projection of Chapter 4 in order to simplify the application of the graph methods of Chapter 3. The operational envelope (OE), necessary to build a graph, is replaced by a projected envelope, which can be of lower dimensionality, easier to define in terms of projected states, or both.

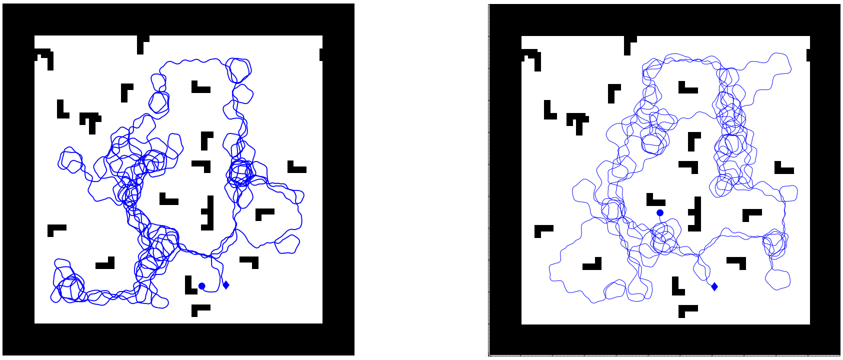


Figure 6.16: Two extracts from the second experimental runs. The dot and the diamond represent initial and final positions. It can be seen how replacing the realization of the model does not substantially alter the behavior of the controlled MAV.

A practical example is provided, showing how a UAV navigation problem can be solved sequentially by iteratively projecting the state.

Furthermore, this chapter introduces Vertex Classification (VC), a method to evaluate policies similar to graph pruning. VC assigns levels and coefficients to the vertices of the graph, which can then be used to estimate and assess the safety of actions, as well as of policies. When compared to pruning, VC is less restrictive towards the obtainable feasible policies, and is less sensitive to the choice in the discretization parameters adopted during graph generation.

The chapter presents then an application of VC to an MAV navigation task, similar to the one of Section 4.5.1. The MAV must navigate within a room, which presents obstacles, and reach a given goal position. The case study shows encouraging and attractive results. The agent, adopting the safest policy of VC, achieves safe flight without collisions, regardless of its simplified beliefs, of the variability of its environment, and of the uncertainty in its prediction model. In particular, applying the safest policy yielded by VC results in trajectories that are analogous to those obtainable by subjecting the MAV to a potential field with repulsive obstacles and attractive goals. Summarizing, VC can prevent undesired transitions, including fatal occurrences, in highly uncertain environments and in presence of limited sensor information. Furthermore, the use of graph methods to discretize and represent policies makes it a viable approach for UAVs with limited computational capabilities. Therefore, VC appears as a possible method to address all three *challenges* of *Safety*, *Robustness*, and *Online efficiency*.

7

Discussion and conclusions

*This chapter summarizes the content of the entire dissertation, and discusses how the methods proposed in the previous chapters address the three main challenges of **safety**, of **online efficiency**, and of **robustness**. Then, the main findings and the final conclusions derived from this research on safe, online, robust RL exploration for UAV agents are presented. Finally, the chapter suggests several fields of continuation and improvement on the topic.*

7.1. Discussion

This dissertation began with stating the three challenges of applying RL techniques to online exploration for UAVs. The *challenge of safety* consists in avoiding harm to the UAV or to the environment during trial-and-error online exploration. The agent has only a limited time to evaluate the safety of its own actions during real-life, online tasks, which constitutes the *challenge of online efficiency*. The *challenge of robustness* means that learning in a safer replica of the environment, either supervised or simulated, is not always reliable due to the discrepancy between the true environment and the replica. The goal of this thesis is then summarized as follows:

> *To investigate the problem of online, safe, robust exploration for aerospace platforms, and to develop potential solutions to the problem of unsafe blind search in accordance with the properties of adaptability, autonomy and model independence of reinforcement learning.*

The objective of this thesis is achieved in two parts. In Part I, three different key approaches are developed. The *Heuristic* methods of Chapter 2 address directly the challenge of safety. The *Graph* methods of Chapter 3 mitigate the challenge of online efficiency. The *Hierarchical* methods of Chapter 4 provide an answer to the challenge of robustness. In Part II, two hybrid methods are devised that aim at combining the three key approaches. The *safety metrics* of Chapter 5 merge the graph-based policies of Chapter 3 with the risk perception and the metrics of Chapter 2. *Vertex Classification*, presented in Chapter 6, adds the hierarchical state projection of Chapter 4 to risk perception and graph-based policies, therefore combining all three key approaches.

7.1.1. Addressing the challenge of safety

Arguably, learning from experience is the most prominent feature of RL. This allows agents to learn optimal and non-trivial policies autonomously, i.e., without guidance from a human operator, and regardless of whether a model of the environment is provided. The inherent consequence of this approach, however, is that the agent must attempt suboptimal actions at least once in order to learn that these should be avoided. This is unacceptable when suboptimality entails actions that can damage the UAV or its surroundings. In order to avoid these *fatal occurrences* during online exploration, the agent must be able to assess the safety of each of its actions. This is the challenge of safety.

Chapter 2 investigates how this assessment can be made by UAV agents. The presence of two prerequisites is assumed. The first prerequisite is *risk perception* in order to discover the set of fatal states, on the assumption that this is time-invariant. The second prerequisite is a *bounding model* which can predict, albeit as an overapproximation, how the state of the environment will change. Suggestions on how to fulfill these two prerequisites for a UAV agent are presented.

The Safety Handling Exploration with Risk Perception Algorithm (SHERPA) propagates a given command, i.e., a sequence of actions, and observes the resulting trajectory overapproximation given by the bounding model. The agent first verifies that this is entirely within the safe region of the state space, as provided by the risk perception. Then, it checks whether the endpoint of the trajectory is within a predetermined distance from any previously visited state, or from any equilibrium point. If the command satisfies these two conditions, i.e., it is a *backup*, the agent is guaranteed not to cause a fatal occurrence during the trajectory; additionally, the more the endpoint is closer to a visited state, the more the agent is likely to find safe actions during the following iteration. The algorithm is validated in simulation on a simplified quadrotor task, and compared to a potential-based method. SHERPA is found to result in a safer and more reasonable behavior than the one obtained with the potential-based method, as well as to be more robust to differences in the UAV model uncertain parameters.

Therefore, SHERPA provides the agent with an assessment procedure to determine the safety of actions, thus answering the challenge of safety. SHERPA does not require the presence of an external operator or of a previously defined safe policy, and it does not require the agent to be reinitialized episodically, using samples of the state trajectory instead. Furthermore, SHERPA can be utilized in combination with any exploratory policy (e.g., with policy search). As a result, SHERPA increases the safety of exploration without penalizing the autonomy and the adaptability of RL methods.

While SHERPA addresses the challenge of safety, the same cannot be said for the challenge of online efficiency. Depending on the complexity of the environment's dynamics, backup assessment can be computationally intensive and potentially unfeasible for UAV applications. This is partially mitigated by OptiSHERPA, also introduced in Chapter 2. This successor algorithm reformulates the safety assessment as a receding horizon optimization, with the addition of the *distance* and *evasion* metrics. OptiSHERPA selects, among a finite set of candidates, the action which optimizes its metric. This algorithm is implemented in simulation on a linearized model of a fighter aircraft. The algorithm is found to constrain the evolution of the system from the equilibrium when implementing the distance metric, and to avoid unsafe regions of the state space when implementing the evasion metric.

7.1.2. Addressing the challenge of online efficiency

Online computational efficiency is an often overlooked requirement of RL algorithms, in favor of other indicators that quantify the optimality of learning, such as convergence time or state space coverage. This tendency can be traced back to earlier applications of the method, e.g., gridworlds [71] and games [122], as well as in many widespread testbeds, such as the Mountain Car problem [123]. However, computational efficiency acquires a primary role in online safe exploration. Before an action can be attempted, the agent must assess its safety, e.g., by looking for backups. If the assessment procedure is not sufficiently fast, the agent might not be able to control the system in real-time, or might have to attempt unscrutinized and possibly fatal actions. This constitutes the challenge of online efficiency.

Chapter 3 addresses this challenge by introducing a graph representation of the bounding model dynamics. An operational envelope (OE) is predefined which represents the desired conditions of the environment. Then, three discretizations are applied in terms of state, of actions, and of time, to obtain the hypergraph in the form of look-up matrices. These allow to overapproximate the reachable set of the environment in a computationally efficient way.

The trajectory predictions needed for safety assessment, e.g., the interval propagations required for backups searches, can be replaced by checking an index of the look-up matrices, which is computationally less intensive. Furthermore, the proposed equispaced tiling adopted for the state discretization significantly reduces the complexity of the graph generation, which is also scalable with the tiling refinement. This allows one to extend the method to OEs that are obtained online, in addition to those that are known a-priori, by adopting a sufficiently coarse refinement. The caveat in this case is that the more a tiling is coarse, the less the obtained trajectory approximations are reliable. Given the above considerations, graph methods are found to mitigate the challenge of online efficiency.

Concerning the challenge of safety, Chapter 3 introduces also *graph pruning*, which is a further approach for safety assessment. This method identifies and prunes the sinks of the graph, i.e., those vertices that are bound to violate the OE. The pruned graph, which does not contain sinks, can be used to assess the safety of actions, in the event that the OE is a subset of the safe state space. Furthermore, this assessment can be extended to whole policies as well.

Graph pruning is tested in simulation on three variants of a simplified UAV navigation task. These confirm that, when the envelope is time-invariant, the method is reliable and actions are safe. In the third variant, however, the UAV presents an envelope that changes during online exploration. As soon as a new envelope is assigned, the previous graph is pruned further to assess safety in the new environment. Afterwards, a new graph is generated and pruned. Both these procedures are shown to be applicable to online exploration when the change in the environment, and thus in the envelope, is sudden but moderate.

7.1.3. Addressing the challenge of robustness

Learning a task off-line in a replica of the environment constitutes an alternative to online learning in the actual environment. This alternative can be considered to be safe if the replica is either a simulation or a real-life artificial or supervised environment modified to prevent dangers. However, policies learned within replicas are likely to suffer from the "reality gap", i.e., to be inefficient or unsafe due to the differences between the replica and the real environment. This constitutes the challenge of robustness, and can result from modeling uncertainties, from misrepresentation of the environment, or from both.

Chapter 4 starts with an investigation of how to apply *temporally extended actions* (TEAs) for safe exploration, and introduces the field of Safe Hierarchical Reinforcement Learning (SHRL). TEAs permit state abstraction, which results in policies that are more robust to uncertainties in the environment, addressing the challenge of robustness. Additionally, they allow to embed designer's knowledge and to restrict the set of discoverable policies, which contributes to mitigate the challenge of safety. The main contribution of Chapter 4 consists in the *Virtual*

Safety Training (VST) strategy. This method allows to train HRL agents off-line in the presence of uncertainties on the environment as well as in the model. First, *state projection* is applied to obtain one projected state that is independent of or relative to the remaining ones. Then, a belief set is created to approximate the projected fatal state space. Learning can then be performed off-line.

The strategy is applied to two very different test cases. The first one is a simulated goal-finding task. The discrete environment consists of a ground robot with deterministic dynamics placed within a maze. Due to the deterministic dynamics, and to the discreteness of the environment, which allows an *exhaustive* belief set, the policy learned during VST is found to be entirely safe, in addition to enhance exploration with respect to other HRL algorithms. The second test case also consists in a goal-finding task within a cluttered environment. A non-exhaustive belief set is obtained by abstracting the sensor reading through *sectors*, while the uncertain dynamics are taken into account by learning a weighted action-value function via Monte Carlo simulations. Even though collisions are observed, the resulting SHRL agent is found to adopt cautious behavior, avoiding densely cluttered areas, and to outperform a flat agent with more extensive and informative exploration.

By applying VST, the SHRL agent learns off-line policies for one or more of its constituent *machine*, *option* or *subtask* (MOS), after which it explores the previously unknown actual environment, in order to learn a hierarchically optimal policy. Depending on the accuracy of the MOS belief set, and on the uncertainty of the training model, it is observed that the UAV agent performs online exploration that is safe, or at least safer than an equivalent flat policy. Thus VST, as part of SHRL, addresses the challenge of robustness.

7.1.4. Addressing multiple challenges: hybrid methods

While the key methods of Part I are found to address the challenges of safety, online efficiency and robustness, the hybrid methods of Part II merge the individual contributions of the key methods.

The first method, introduced in **Chapter 5**, rests on three elements: a warning function, a graph representation of the dynamics, and one or more *safety metrics*. The warning function, which is a form of risk perception, and the graph formulation derive respectively from Chapters 2 and 3, while the safety metrics represent an original contribution of the chapter. These are designed to accommodate the tile discretization of the state and the trajectory prediction via graphs. The *operative metric* (OM) assigns weights to tiles depending on the return of the warning function and on whether they have been visited. The *proximity metric* (PM) can instead be seen as a tile equivalent of the distance metric of Chapter 2. The action of the agent is selected via a receding horizon optimization, where the optimality is given by either the OM or the PM.

Two tasks are simulated to test the metrics. The first task consists of flying a UAV in an indoor environment with no obstacles for a set amount of time. The agent utilizing the OM manages to complete all simulations without colliding, while at the same time exploring the room. Depending on the initial conditions, the agent with the PM is found to execute iterative maneuvers, e.g., constant turns, which are safe but constrain the exploration. In the majority of episodes, however, the PM does not manage to prevent collisions in this task. The second task entails controlling a fighter aircraft, whose model is linearized, via elevator deflection. The PM is found again to constrain the evolution of the system by keeping the flight angle angle approximately zero, which results in longer episode durations when compared to the OM. Summarizing, the OM is observed to be safer when the prediction horizon is sufficient enough to account for the insurgence of risks; otherwise, the restriction imposed by the PM is found to provide safer exploration.

The use of safety metrics combines the contributions of Chapters 2 and 3. The use of a warning function (which is a form of risk perception) provides a way to identify and avoid fatal states. Additionally, the PM increases safety of exploration in the presence of *lead-to-fatal* states, similarly to the distance metric of Chapter 2. The metrics contribute thus to address the challenge of safety. Due to the similarity between the two elevator deflection tasks of Chapters 2 and 5, it is possible that the introduction of a reactive safety metric, akin to the evasion metric of Chapter 2, might reduce fatal occurrences. Additionally, the graph formulation of the dynamics and of the metrics, inherited from Chapter 3, guarantees a low and a-priori known computational complexity of the action selection, thus addressing the challenge of online efficiency.

Chapter 6 introduces *Vertex Classification* (VC), a hybrid method which takes contributions from all key approaches. In order to apply VC, several steps are necessary. First, the state is projected into a representation that is independent of or relative to the other projected states, as detailed in Chapter 4. Then, a subset of the projected space is selected as the OE of the agent, and is partitioned into a tiling in order to obtain a graph representation of the projected dynamics, in accordance with Chapter 3. During graph generation, actions that violate the envelope constraints are identified as critical transitions, and classified in terms of *desirability*. Desired transitions are those that contribute to reach the goal of the task, e.g., to reach a goal destination for the agent, while undesired transitions are those that increase the chance of a fatal occurrence, according to risk perception of Chapter 2. Then, vertices are assigned a *level* and a *coefficient*. After assigning an *intensity* to each critical transition, levels and coefficients can be combined into a final weight quantifying the safety of state-action couples.

VC has major resemblances with its predecessor method, graph pruning, in that it can be used to evaluate RL policies, e.g., by preventing actions whose safety is below a given threshold. However, it is less sensitive to discretization parameters. Additionally, VC always individuates a *safest policy*. The method is applied in simulation to an MAV navigation task, essentially the same presented in Chapter 4. The sector abstraction is applied again in order to define the desirability of critical transitions. All episodes generated by the safest policy are found to be collision free, regardless of environment, state initialization and model realization. Analogies are observed between the behavior of the agent implementing the safest policy and what could be expected from the presence of a potential field exerting forces on the MAV. For example, it is observed that the agent can get stuck in what appear to be local minima, due to the attraction of the goal and to the repulsion of the obstacles.

Results show that VC is a possible solution to the three challenges of safety, of robustness, and of online efficiency. Fatal occurrences, observed through risk perception and accounted for in the belief set, are avoided through the repulsive forces generated by undesired transitions, thus addressing the challenge of safety during online exploration. State projection is able to account for the uncertainty in the environment and in the model, as shown by different episodes of the MAV task, therefore answering the challenge of robustness. Finally, the graph formulation of the dynamics, and the assignment of weights to vertices, allows to assess safety for the current policy without propagating the dynamics online. Due to its low computational complexity, VC can account for the challenge of online efficiency as well.

7.2. Final conclusions

With respect to the goal of safe, online, robust RL exploration, and to the three main challenges to the goal, the following conclusions are drawn based on the methodologies and on the results of this dissertation.

On the challenge of safety

1. In order to prevent fatal occurrences, an agent must be able to perceive whether or not a neighborhood of the current state of the environment intersects the fatal state space.
2. In order to prevent fatal occurrences, an agent must be able to predict or overestimate the state trajectory corresponding to a given action sequence.
3. Safe policies can be discovered autonomously and online by the agent in the form of *backups*, in absence of a-priori known ones.
4. Utilizing a receding horizon optimization to define the policy of the agent, with a metric penalizing deviation from visited states, has the effect of constraining the evolution of the environment.

On the challenge of online efficiency

5. Precomputing an overapproximation of state transitions within a graph formulation makes safety assessment feasible for online exploration.
6. A policy compatible with a pruned graph is guaranteed to be feasible with respect to the given operational envelope, in the event that this envelope is time-invariant.

On the challenge of robustness

7. Safe policies learned off-line through state abstraction are likely to be safe when applied to uncertain environments, as long as the uncertainty is also accounted for in the abstraction.

8. Policies of machines, options or subtasks, that are learned off-line in a projected state space and via state abstraction, prevent initial blind search for a Hierarchical Reinforcement Learning agent.
9. Safe online exploration is possible, if primitive actions are executed only by a machine, option or subtask for which a safe policy is learned off-line, depending on the exhaustiveness of the belief set and on the uncertainty of the model.

On safe online robust exploration

10. An unmanned aerial vehicle reinforcement learning agent can avoid unsafe blind search in unknown fatal environments, by initializing its policy off-line via state projection and state abstraction, in combination with a computationally efficient representation of the environment and a belief set of fatal occurrences based on risk perception.

7.3. Recommendations and future work

This section presents recommendations for the UAV and machine learning communities on future work concerning safe exploration for UAVs, in light of the limitations and simplifications adopted throughout this dissertation. Finally, the societal impact of this dissertation is briefly discussed.

For the UAV community

All the methods presented in this dissertation are tested and validated exclusively through simulations. As a result, sensor noise and environmental effects are not considered. Modeling such additional elements and in general improving the quality of the validation models would constitute a step towards a more realistic implementation of the methods, as would be an application of these on real-life UAVs.

The theoretical background of this dissertation entails a general and comprehensive definition of fatal occurrences. These consist of those state transitions that affect negatively the exploration, and that cannot be compensated by any amount of later reward. However, in the presented applications, fatal occurrences are almost consistently limited to collisions. While these are arguably the main safety-critical concerns of a UAV agent, other safety or mission-critical occurrences, such as battery depletion or intrusion in restricted airspace, could be devised when considering more complex UAV tasks. The investigation of these occurrences would benefit the applicability of the proposed methods.

As a related topic, how to obtain risk perception depends on the specific fatal occurrences of a given task; suggestions are presented in Chapter 2. In the event that fatal occurrences are limited to collisions, risk perception can be provided by proximity sensors, such as sonars. However, more complex occurrences would require more refined sensors, or combinations of sensors, in order to obtain risk perception. Addressing this necessity, by investigating such sensors and sensor combinations, would also contribute to the applicability of the method.

For the machine learning community

Several methods presented in this dissertation alter or constrain the policy of the UAV agent. For example, SHERPA acts as a filter on the exploratory action, while VC assigns a degree of safety to actions depending on the predicted vertex transition. These restrictions are not integrated within the RL framework. While this is partly a design choice, in order to avoid conflicting rewards, it also makes it difficult to validate the convergence and optimality of the learned policy. Also, this dissertation does not examine how a policy learned with one of the proposed methods can be used independently after convergence. Investigating how to guarantee convergence and independence of the agent's policy appears as a promising and multidisciplinary topic that could improve the impact of this dissertation's methods.

One of the core methodologies of this dissertation is Interval Analysis [88] (IA). With this method, uncertainties can be represented, propagated and estimated in a simple, efficient, and reliable way. However, IA does not yield probabilities for specific state transitions within the overestimation of the state trajectory, i.e., all the estimated trajectories are considered equiprobable. Furthermore, the dependency problem means that predicted trajectories tend to overapproximate the actual trajectories more poorly with the increase of integration time. Considering the limitations above, an investigation of different representations and propagation of uncertainties is likely to improve the applicability of the proposed methods. Finally, a different uncertainty representation could increase the efficacy of these methods with respect to the challenge of robustness.

Societal impact of this dissertation

Automation constitutes a driving innovation in aerospace. In particular, UAVs are foreseen to play an important role in future airspace, and to be increasingly employed in several applications, from surveillance to deliveries. However, automation for UAVs also represents a challenge for the aerospace community, because of the effort required to design autonomous controllers, and due to the difficulty in obtaining high-fidelity models of these platforms. Among other machine learning methods, RL can be instrumental in overcoming this challenge, making it possible to discover optimal policies in an autonomous, adaptable and robust way.

However, RL does not come without its own challenges when UAV applications are considered: its trial-and-error nature must be accounted for when considering environments in which exploration might cause damage to the UAV, to its surroundings, or to bystanders. In this dissertation, the challenges of safety, of online efficiency, and of robustness are individuated and addressed with multiple approaches, in order to increase the applicability of RL in online tasks. Thus, the proposed methods constitute a step towards enabling the use of RL agents for UAVs, and therefore to make aerospace safer, more autonomous, and more efficient.

References

[1] N. V. Hoffer, C. Coopmans, A. M. Jensen, and Y. Chen, *A survey and categorization of small low-cost unmanned aerial vehicle system identification,* Journal of Intelligent & Robotic Systems **74**, 129 (2014).

[2] H. E. Taha, M. R. Hajj, and A. H. Nayfeh, *Flight dynamics and control of flapping-wing mavs: a review,* Nonlinear Dynamics **70**, 907 (2012).

[3] C. T. Orlowski and A. R. Girard, *Dynamics, stability, and control analyses of flapping wing micro-air vehicles,* Progress in Aerospace Sciences **51**, 18 (2012).

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

[5] R. Bellman and R. E. Kalaba, *Dynamic programming and modern control theory*, Vol. 81 (Citeseer, 1965).

[6] C. J. Watkins and P. Dayan, *Q-learning,* Machine learning **8**, 279 (1992).

[7] G. A. Rummery and M. Niranjan, *On-line Q-learning using connectionist systems* (University of Cambridge, Department of Engineering, 1994).

[8] R. Parr and S. Russell, *Reinforcement learning with hierarchies of machines,* Advances in neural information processing systems , 1043 (1998).

[9] L. Orseau and S. Armstrong, *Safely interruptible agents,* in *Uncertainty in Artificial Intelligence: 32nd Conference (UAI 2016), edited by Alexander Ihler and Dominik Janzing* (2016) pp. 557–566.

[10] S. M. LaValle, *Planning Algorithms* (Cambridge University Press, Cambridge, U.K., 2006) available at http://planning.cs.uiuc.edu/.

[11] M. Yogeswaran and S. Ponnambalam, *Reinforcement learning: exploration–exploitation dilemma in multi-agent foraging task,* Opsearch **49**, 223 (2012).

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

[13] J. García and F. Fernández, *A comprehensive survey on safe reinforcement learning,* J. Mach. Learn. Res. **16**, 1437 (2015).

[14] P. Thomas, G. Theocharous, and M. Ghavamzadeh, *High confidence policy improvement,* in *Proc. of the 32nd Int. Conf. Mach. Learn. (ICML-15)*, edited by D. Blei and F. Bach (JMLR Workshop and Conference Proceedings, 2015) pp. 2380–2388.

[15] R. Neuneier and O. Mihatsch, *Risk sensitive reinforcement learning,* in *Proceedings of the 11th International Conference on Neural Information Processing Systems*, NIPS'98 (MIT Press, Cambridge, MA, USA, 1998) pp. 1031–1037.

[16] O. Mihatsch and R. Neuneier, *Risk-sensitive reinforcement learning,* Machine Learning **49**, 267 (2002).

[17] Y. Shen, M. J. Tobia, T. Sommer, and K. Obermayer, *Risk-sensitive reinforcement learning,* Neural Computation **26**, 1298 (2014).

[18] P. Geibel and F. Wysotzki, *Risk-sensitive reinforcement learning applied to control under constraints,* J. Artif. Intell. Res. **24**, 81 (2005).

[19] S. P. Coraluppi and S. I. Marcus, *Risk-sensitive and minimax control of discrete-time, finite-state markov decision processes,* Automatica **35**, 301 (1999).

[20] K. Driessens and S. Džeroski, *Integrating guidance into relational reinforcement learning,* Machine Learning **57**, 271 (2004).

[21] B. D. Argall, S. Chernova, M. Veloso, and B. Browning, *A survey of robot learning from demonstration,* Robotics and Autonomous Systems **57**, 469 (2009).

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

[23] C. Gehring and D. Precup, *Smart exploration in reinforcement learning using absolute temporal difference errors,* in *Proc. 2013 Int. Conf. Autonomous Agents Multi-agent Syst.* (International Foundation for Autonomous Agents and Multiagent Systems, 2013) pp. 1037–1044.

[24] A. Hans, D. Schneegaß, A. M. Schäfer, and S. Udluft, *Safe exploration for reinforcement learning.* in *ESANN* (2008) pp. 143–148.

[25] M. Pecka, K. Zimmermann, and T. Svoboda, *Safe exploration for reinforcement learning in real unstructured environments,* in *Proc. of the Computer Vision Winter Workshop* (2015).

[26] F. J. G. Polo and F. F. Rebollo, *Safe reinforcement learning in high-risk tasks through policy improvement,* in *Adaptive Dynamic Programming And Reinforcement Learning (ADPRL), 2011 IEEE Symposium on* (2011) pp. 76–83.

[27] T. M. Moldovan and P. Abbeel, *Safe Exploration in Markov Decision Processes* *,* in *Proc. 29th Int. Conf. Mach. Learn.* (icml.cc / Omnipress, 2012) p. 188.

[28] D. P. Bertsekas and J. N. Tsitsiklis, *Neuro-dynamic programming: an overview,* in *Proceedings of 1995 34th IEEE Conference on Decision and Control*, Vol. 1 (1995) pp. 560–564 vol.1.

[29] D. Ernst, P. Geurts, and L. Wehenkel, *Tree-based batch mode reinforcement learning,* Journal of Machine Learning Research **6**, 503 (2005).

[30] Y. Zhu and D. Zhao, *A data-based online reinforcement learning algorithm with high-efficient exploration,* in *2014 IEEE Symposium on Adaptive Dynamic Programming and Reinforcement Learning (ADPRL)* (2014) pp. 1–6.

[31] R. I. Brafman and M. Tennenholtz, *R-max - a general polynomial time algorithm for near-optimal reinforcement learning,* J. Mach. Learn. Res. **3**, 213 (2003).

[32] L. P. Kaelbling, M. L. Littman, and A. W. Moore, *Reinforcement learning: A survey,* Journal of artificial intelligence research **4**, 237 (1996).

[33] M. Abramson and H. Wechsler, *Tabu search exploration for on-policy reinforcement learning,* in *Proceedings of the International Joint Conference on Neural Networks, 2003.*, Vol. 4 (2003) pp. 2910–2915 vol.4.

[34] J. A. Bagnell, S. Kakade, A. Y. Ng, and J. Schneider, *Policy search by dynamic programming,* in *Proceedings of the 16th International Conference on Neural Information Processing Systems*, NIPS'03 (MIT Press, Cambridge, MA, USA, 2003) pp. 831–838.

[35] A. Y. Ng and M. Jordan, *Pegasus: A policy search method for large mdps and pomdps,* in *Proceedings of the Sixteenth Conference on Uncertainty in Artificial Intelligence*, UAI'00 (Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2000) pp. 406–415.

[36] J. Kober and J. Peters, *Policy search for motor primitives in robotics,* in *Proceedings of the 21st International Conference on Neural Information Processing Systems*, NIPS'08 (Curran Associates Inc., USA, 2008) pp. 849–856.

[37] J. A. Bagnell and J. G. Schneider, *Autonomous helicopter control using reinforcement learning policy search methods,* in *Proceedings 2001 ICRA. IEEE International Conference on Robotics and Automation (Cat. No.01CH37164)*, Vol. 2 (2001) pp. 1615–1620 vol.2.

[38] T. Degris, P. M. Pilarski, and R. S. Sutton, *Model-free reinforcement learning with continuous action in practice,* in *2012 American Control Conference (ACC)* (2012) pp. 2177–2182.

[39] T. Hanselmann, L. Noakes, and A. Zaknich, *Continuous-time adaptive critics,* IEEE Transactions on Neural Networks **18**, 631 (2007).

[40] T. Hanselmann, L. Noakes, and A. Zaknich, *Continuous adaptive critic designs,* in *Proceedings. 2005 IEEE International Joint Conference on Neural Networks, 2005.*, Vol. 5 (2005) pp. 3001–3006 vol. 5.

[41] G. Zames, *Feedback and optimal sensitivity: Model reference transformations, multiplicative seminorms, and approximate inverses,* IEEE Transactions on Automatic Control **26**, 301 (1981).

[42] A. Al-Tamimi, F. L. Lewis, and M. Abu-Khalaf, *Model-free q-learning designs for linear discrete-time zero-sum games with application to h-infinity control,* Automatica **43**, 473 (2007).

[43] B. Kiumarsi, F. L. Lewis, and Z.-P. Jiang, *Control of linear discrete-time systems: Off-policy reinforcement learning,* Automatica **78**, 144 (2017).

[44] B. Luo, H. N. Wu, and T. Huang, *Off-policy reinforcement learning for h_∞ control design,* IEEE Transactions on Cybernetics **45**, 65 (2015).

[45] X. Yang, D. Liu, and Q. Wei, *Data-based robust control for unknown nonlinear systems,* in *2016 35th Chinese Control Conference (CCC)* (2016) pp. 3123–3128.

[46] B. Luo, H. Wu, T. Huang, and D. Liu, *Data-based approximate policy iteration for affine nonlinear continuous-time optimal control design,* Automatica **50**, 3281 (2014).

[47] Y. Jiang and Z. P. Jiang, *Robust adaptive dynamic programming and feedback stabilization of nonlinear systems,* IEEE Transactions on Neural Networks and Learning Systems **25**, 882 (2014).

[48] Z.-P. Jiang and Y. Jiang, *Robust adaptive dynamic programming for linear and nonlinear systems: An overview,* European Journal of Control **19**, 417 (2013), the Path of Control.

[49] H. Fujita and S. Ishii, *Model-based reinforcement learning for partially observable games with sampling-based state estimation,* Neural Computation **19**, 3051 (2007).

[50] M. Chen, E. Frazzoli, D. Hsu, and W. S. Lee, *Pomdp-lite for robust robot planning under uncertainty,* in *2016 IEEE International Conference on Robotics and Automation (ICRA)* (2016) pp. 5427–5433.

[51] S. Devlin, M. Grzes, and D. Kudenko, *Reinforcement learning in robocup keepaway with partial observability,* in *2009 IEEE/WIC/ACM International Joint Conference on Web Intelligence and Intelligent Agent Technology*, Vol. 2 (2009) pp. 201–208.

[52] M. Deisenroth and C. E. Rasmussen, *Pilco: A model-based and data-efficient approach to policy search,* in *Proceedings of the 28th International Conference on machine learning (ICML-11)* (2011) pp. 465–472.

[53] P. Poupart and N. A. Vlassis, *Model-based bayesian reinforcement learning in partially observable domains,* in *10th International Symposium on Artificial Intelligence and Mathematics (ISAIM)* (2008).

[54] M. Strens, *A bayesian framework for reinforcement learning,* in *In Proceedings of the Seventeenth International Conference on Machine Learning* (ICML, 2000) pp. 943–950.

[55] D. E. Koulouriotis and A. Xanthopoulos, *Reinforcement learning and evolutionary algorithms for non-stationary multi-armed bandit problems,* Applied Mathematics and Computation **196**, 913 (2008).

[56] M. Abdoos, N. Mozayani, and A. L. Bazzan, *Traffic light control in non-stationary environments based on multi agent q-learning,* in *Intelligent Transportation Systems (ITSC), 2011 14th International IEEE Conference on* (IEEE, 2011) pp. 1580–1585.

[57] A. Marinescu, I. Dusparic, A. Taylor, V. Cahill, and S. Clarke, *Decentralised multi-agent reinforcement learning for dynamic and uncertain environments,* CoRR **abs/1409.4561** (2014).

[58] M. A. K. Jaradat, M. Al-Rousan, and L. Quadan, *Reinforcement based mobile robot navigation in dynamic environment,* Robotics and Computer-Integrated Manufacturing **27**, 135 (2011).

[59] E. D. S. Costa and M. M. Gouvea Jr, *Autonomous navigation in dynamic environments with reinforcement learning and heuristic,* in *Machine Learning and Applications (ICMLA), 2010 Ninth International Conference on* (IEEE, 2010) pp. 37–42.

[60] M. Pieters and M. A. Wiering, *Q-learning with experience replay in a dynamic environment,* in *2016 IEEE Symposium Series on Computational Intelligence (SSCI)* (2016) pp. 1–8.

[61] R. Zhuo, Z.-h. CHEN, and C.-l. CHEN, *Navigation for mobile robots using reinforcement learning and fuzzy logic,* Computer Simulation **8**, 043 (2005).

[62] Z. Wang, C. Chen, H.-X. Li, D. Dong, and T.-J. Tarn, *A novel incremental learning scheme for reinforcement learning in dynamic environments,* in *Intelligent Control and Automation (WCICA), 2016 12th World Congress on* (IEEE, 2016) pp. 2426–2431.

[63] S. Singh, R. L. Lewis, A. G. Barto, and J. Sorg, *Intrinsically motivated reinforcement learning: An evolutionary perspective,* IEEE Transactions on Autonomous Mental Development **2**, 70 (2010).

[64] T. Y. Tang, S. Egerton, and N. Kubota, *Reinforcement learning in non-stationary environments: An intrinsically motivated stress based memory retrieval performance (sbmrp) model,* in *Fuzzy Systems (FUZZ-IEEE), 2014 IEEE International Conference on* (IEEE, 2014) pp. 1728–1735.

[65] M. A. Wiering, *Model-based reinforcement learning in dynamic environments,* Department of Information and Computing Sciences, Utrecht University, Tech. Rep. UU-CS-2002-029 (2002).

[66] M. A. Wiering, *Reinforcement learning in dynamic environments using instantiated information.* in *ICML* (2001) pp. 585–592.

[67] A. G. Barto and S. Mahadevan, *Recent advances in hierarchical reinforcement learning,* Discrete Event Dynamic Systems **13**, 341 (2003).

[68] H. Kwakernaak, *Robust control and h∞-optimization - tutorial paper,* Automatica **29**, 255 (1993).

[69] E. van Kampen, Q. Chu, and J. Mulder, *Continuous adaptive critic flight control aided with approximated plant dynamics,* in *AIAA Guidance, Navigation, and Control Conference and Exhibit* (2006).

[70] S. Ferrari and R. F. Stengel, *Online adaptive critic flight control,* J. Guid. Control Dyn. **27**, 777 (2004).

[71] R. S. Sutton, *Generalization in reinforcement learning: Successful examples using sparse coarse coding,* Advances in neural information processing systems , 1038 (1996).

[72] D. Haussler, *Probably approximately correct learning,* in *Proc. of the 8th Nat. Conf. Artif. Intell. - Vol. 2,* AAAI'90 (AAAI Press, 1990) pp. 1101–1108.

[73] D. Zhao and Y. Zhu, *Mec—a near-optimal online reinforcement learning algorithm for continuous deterministic systems,* IEEE Trans. Neural Netw. Learn. Syst. **26**, 346 (2015).

[74] D. P. Bertsekas and J. N. Tsitsiklis, *Neuro-Dynamic Programming*, 1st ed. (Athena Scientific, 1996).

[75] J. Si and Y. Wang, *Online learning control by association and reinforcement,* IEEE Trans. Neural Netw. **12**, 264 (2001).

[76] R. Bellman, *Dynamic Programming*, 1st ed. (Princeton University Press, Princeton, NJ, USA, 1957).

[77] M. C. Choy, D. Srinivasan, and R. L. Cheu, *Neural networks for continuous online learning and control,* IEEE Trans. Neural Netw. **17**, 1511 (2006).

[78] Y. J. Liu, L. Tang, S. 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. **26**, 165 (2015).

[79] 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. **24**, 16 (2016).

[80] 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. **45**, 1372 (2015).

[81] 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. **15**, 639 (2004).

[82] 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. **26**, 2301 (2015).

[83] 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. **26**, 1035 (2015).

[84] M. Pecka and T. Svoboda, *Modelling and simulation for autonomous systems: First international workshop, mesas 2014, rome, italy, may 5-6, 2014, revised selected papers,* (Springer International Publishing, Cham, 2014) Chap. Safe Exploration Techniques for Reinforcement Learning – An Overview, pp. 357–375.

[85] C. Szepesvári, *Algorithms for reinforcement learning,* Synthesis Lectures Artif. Intell. Mach. Learn. **4**, 1 (2010).

[86] P. Geibel, *Reinforcement learning with bounded risk,* in *ICML* (2001) pp. 162–169.

[87] T. Fraichard and H. Asama, *Inevitable collision states — a step towards safer robots?* Advanced Robotics **18**, 1001 (2004).

[88] R. E. Moore, *Interval analysis*, Vol. 4 (Prentice-Hall Englewood Cliffs, 1966).

[89] O. Khatib, *Real-time obstacle avoidance for manipulators and mobile robots,* Int. J. Robot. Res. **5**, 90 (1986).

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

[91] 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, 2003.*, Vol. 3 (IEEE, 2003) pp. 2926–2931.

[92] V. J. Lumelsky and A. A. Stepanov, *Path-planning strategies for a point mobile automaton moving amidst unknown obstacles of arbitrary shape,* Algorithmica **2**, 403 (1987).

[93] K. P. Tee, S. S. Ge, and E. H. Tay, *Barrier lyapunov functions for the control of output-constrained nonlinear systems,* Automatica **45**, 918 (2009).

[94] P. Wieland and F. Allgöwer, *Constructive safety using control barrier functions,* IFAC Proceedings Volumes **40**, 462 (2007).

[95] M. Z. Romdlony and B. Jayawardhana, *Uniting control lyapunov and control barrier functions,* in *53rd IEEE Conf. Decision Control* (IEEE, 2014) pp. 2293–2298.

[96] 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 **39**, 1316 (2009).

[97] R. J. Wilson, *An introduction to graph theory* (Pearson Education India, 1970).

[98] J. Garcke and I. Klompmaker, *Adaptive sparse grids in reinforcement learning,* in *Extraction of Quantifiable Information from Complex Systems* (Springer, 2014) pp. 179–194.

[99] A. Feuer and M. Heymann, *ω-invariance in control systems with bounded controls,* Journal of mathematical Analysis and Applications **53**, 266 (1976).

[100] M. Oishi, I. Mitchell, C. Tomlin, and P. Saint-Pierre, *Computing viable sets and reachable sets to design feedback linearizing control laws under saturation,* in *Decision and Control, 2006 45th IEEE Conference on* (IEEE, 2006) pp. 3801–3807.

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

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

[103] H. Kwatny and R. Allen, *Safe set maneuverability of impaired aircraft,* in *AIAA Atmospheric Flight Mechanics Conference* (2012) p. 4405.

[104] A. G. Barto and S. Mahadevan, *Recent advances in hierarchical reinforcement learning,* Discrete Event Dynamic Systems **13**, 341 (2003).

[105] R. S. Sutton, D. Precup, and S. Singh, *Between mdps and semi-mdps: A framework for temporal abstraction in reinforcement learning,* Artificial intelligence **112**, 181 (1999).

[106] D. Precup, *Temporal Abstraction in Reinforcement Learning*, Ph.D. thesis (2000).

[107] M. Stolle and D. Precup, *Learning options in reinforcement learning,* in *International Symposium on Abstraction, Reformulation, and Approximation* (Springer, 2002) pp. 212–223.

[108] K. Subramanian, C. Isbell, and A. Thomaz, *Learning options through human interaction,* in *2011 IJCAI Workshop on Agents Learning Interactively from Human Teachers (ALIHT)* (Citeseer, 2011).

[109] E. Brunskill and L. Li, *Pac-inspired option discovery in lifelong reinforcement learning.* in *ICML* (2014) pp. 316–324.

[110] T. G. Dietterich, *Hierarchical reinforcement learning with the maxq value function decomposition,* J. Artif. Intell. Res.(JAIR) **13**, 227 (2000).

[111] P. Dayan and G. E. Hinton, *Feudal reinforcement learning,* in *Advances in neural information processing systems* (Morgan Kaufmann Publishers, 1993) pp. 271–271.

[112] R. Brooks, *A robust layered control system for a mobile robot,* IEEE journal on robotics and automation **2**, 14 (1986).

[113] A. Stout, G. D. Konidaris, and A. G. Barto, *Intrinsically motivated reinforcement learning: A promising framework for developmental robot learning,* Tech. Rep. (DTIC Document, 2005).

[114] C. Chen, H. X. Li, and D. Dong, *Hybrid control for robot navigation - a hierarchical q-learning algorithm,* IEEE Robotics & Automation Magazine **15** (2008).

[115] B. Doroodgar and G. Nejat, *A hierarchical reinforcement learning based control architecture for semi-autonomous rescue robots in cluttered environments,* in *Automation Science and Engineering (CASE), 2010 IEEE Conference on* (IEEE, 2010) pp. 948–953.

[116] H. Moravec and A. Elfes, *High resolution maps from wide angle sonar,* in *Robotics and Automation. Proceedings. 1985 IEEE International Conference on*, Vol. 2 (IEEE, 1985) pp. 116–121.

[117] S. Thrun and A. Bücken, *Integrating grid-based and topological maps for mobile robot navigation,* in *Proceedings of the National Conference on Artificial Intelligence* (1996) pp. 944–951.

[118] B. Yamauchi, *A frontier-based approach for autonomous exploration,* in *Computational Intelligence in Robotics and Automation, 1997. CIRA'97., Proceedings., 1997 IEEE International Symposium on* (IEEE, 1997) pp. 146–151.

[119] Y. Zhou, E. van Kampen, and Q. Chu, *Autonomous navigation in partially observable environments using hierarchical q-learning,* in *International Micro Air Vehicle Competition and Conference 2016*, edited by P. Z. PENG and D. F. LIN (Beijing, PR of China, 2016) pp. 70–76.

[120] J. C. Santamaría, R. S. Sutton, and A. Ram, *Experiments with reinforcement learning in problems with continuous state and action spaces,* Adaptive behavior **6**, 163 (1997).

[121] C. E. Garcia, D. M. Prett, and M. Morari, *Model predictive control: theory and practice—a survey,* Automatica **25**, 335 (1989).

[122] G. Tesauro, *Temporal difference learning and td-gammon,* Communications of the ACM **38**, 58 (1995).

[123] J. A. Boyan and A. W. Moore, *Generalization in reinforcement learning: Safely approximating the value function,* Advances in neural information processing systems , 369 (1995).

Samenvatting

Safe Online Robust Exploration for Reinforcement Learning Control

of Unmanned Aerial Vehicles

Tommaso **Mannucci**

De luchtvaart is recentelijk getuige geweest van een ongekende toename in de interesse voor onbemande luchtvaartuigen (UAVs). Met de ontwikkelingen op het gebied van miniaturisatie en goedkope hardware, variërend van printplaten tot sensoren, wordt de productie van UAVs steeds goedkoper en worden prestaties en vliegtijd van UAVs almaar groter. Als gevolg hiervan hebben "drones" hun intree gemaakt in de hobby-markt als een betaalbaar speelgoed en betrouwbaar werktuig. Bovendien doen verschillende bedrijven onderzoek naar het inzetten van UAVs voor kosteneffectieve diensten, zoals distributie en bezorging.

Daarentegen zijn regelsystemen voor UAVs nog steeds gebaseerd op de klassieke regeltheorie, zoals PID's en robuuste regeltheorie. Deze regelsystemen zijn betrouwbaar bij modelonzekerheden, welke vrij gebruikelijk zijn bij UAVs en in het bijzonder bij micro onbemande luchtvaartuigen (MAVs). Echter, een praktisch nadeel van deze technieken is dat het afstemmen, testen en modeleren tijdens de ontwerpfase veel tijd in beslag nemen. Het vooruitzicht op geheel autonome UAV taken, zonder supervisie van een menselijke piloot, behelst een uitdaging voor deze klassieke regelsystemen en zal het ontwerp van het regelsysteem waarschijnlijk significant vermoeilijken.

Vanuit dit oogpunt heeft reinforcement learning (RL) de potentie om deze uitdagingen aan te kunnen. RL is een type kunstmatige intelligentie die biologische leerprocessen nabootst: een *RL-entiteit* heeft interactie met zijn *omgeving* door het uitvoeren van *acties*, waarna een *beloning* ontvangen wordt. Dit geeft een direct terugkoppeling over hoe goed de actie was, op basis van de door de ontwerper gegeven beloningsfunctie. Het doel van de RL-entiteit is om de toekomstige verdisconteerde beloning te maximaliseren, wat resulteert in een optimaal *beleid*. De kracht van deze *exploratie-procedure* is dat de RL-entiteit autonoom en onafhankelijk van een model kan leren.

Voor luchtvaarttoepassingen is het cruciaal dat deze RL-exploratie veilig is. De RL-entiteit moet onveilige acties, zoals bijvoorbeeld de acties die resulteren in een

botsing, kunnen identificeren zonder dat deze acties daadwerkelijk uitgevoerd worden. Dit is de *Uitdaging van veiligheid*. Het is in principe mogelijk om veiligheid te garanderen door te leren in een veilige of gesimuleerde replica van de daadwerkelijke omgeving, waarin onveilige acties toegestaan zijn; echter, het beleid dat op deze wijze aangeleerd wordt kan onveilig zijn in de echte omgeving als er onzekerheden of verschillen zitten in de replica van de omgeving. Dit vormt de *uitdaging van robuustheid*. Als gegeven is dat de veiligheid van een actie online gëevalueerd kan worden, dan moet deze evaluatie niet teveel rekenkracht vereisen, zodat de agent dit tijdens het aansturen van de UAV kan doen. Dit is de *uitdaging van rekenkracht*.

Het is duidelijk dat deze drie uitdagingen overkomen moeten worden voordat UAVs en MAVs volledig voordeel kunnen hebben van RL. Het doel van dit proefschrift is om deze problemen omtrent de veiligheid, robuustheid en rekenkracht van online RL-exploratie voor UAVs te onderzoeken en om mogelijke oplossingen te ontwikkelen in overeenstemming met de RL-eigenschappen van aanpasbaarheid, autonomie en modelonafhankelijkheid.

Veiligheid is de eerste en met afstand de meest dwingende uitdaging voor een RL-entiteit. Om het probleem te versimpelen wordt gesteld dat onveilige acties die acties zijn die ervoor zorgen dat de omgeving overgaat in een element van de set van fatale toestanden, welke onbekend maar tijd-invariant wordt verondersteld. In afwezigheid van een vooraf bekend veilig beleid, of van een menselijke leraar, zijn er twee competenties vereist om fatale transities te voorkomen. De eerste is *perceptie van risico*, die de vorm aanneemt van een extra terugkoppelingssignaal van de omgeving naar de RL-entiteit en welke de RL-entiteit informeert of er een element van de set van fatale toestanden binnen een vooraf gedefinieerde afstand van de huidige toestand is. De tweede is de beschikbaarheid van een *begrenzend model* welke een conservatieve schatting maakt van de toekomstige overgang van de omgeving gegeven de acties van de RL-entiteit.

Deze twee competenties vormen de centrale strategie achter het *Veilige Exploratie door Perceptie van Risico* (SHERPA) algoritme dat ontwikkeld is in dit proefschrift. Dit is een "veiligheidsfilter" dat tussen de RL-entiteit en de omgeving wordt geplaatst. SHERPA staat alleen acties toe die gegarandeerd niet leiden tot fatale transities en die opgevolgd kunnen worden door *back-up*, een set van acties die transities naar veilige toestanden waarborgen in de buurt van voorheen bezochte toestanden. In het geval dat de actie die de RL-entiteit voorstelt geweigerd wordt door SHERPA, wordt de RL-entiteit gevraagd een nieuwe actie aan te dragen, net zolang tot SHERPA de actie goedkeurt of totdat een tijdslimiet overschreden is, waarop SHERPA de vooraf goedgekeurde back-up uit zal voeren. Hiermee biedt SHERPA de RL-entiteit een autonome veiligheidsbeoordeling gebaseerd op online ervaring. SHERPA is gevalideerd op een versimpeld quadrotor vliegtaak, waaruit blijkt dat SHERPA effectiever is dan een concurrerende methode.

Dit proefschrift ontwikkelt de SHERPA strategie tevens verder uit tot OptiSHERPA. Dit algoritme zet de veiligheidsbeoordeling om in een optimalisatieprobleem door het selecteren van de veiligste actie uit een beperkte set van kandidaten.

Bovendien wordt een ontwijk-maat toegevoegd, zodat de RL-entiteit de actie kan kiezen die het minst aannemelijk is om tot een fatale transitie te leiden. OptiSHERPA is getest in een besturingstaak van een lineair model van een gevechtsvliegtuig.

De tweede uitdaging van *rekenkracht* wordt geadresseerd in dit proefschrift door het gebruik van *grafen* tijdens de veiligheidsbeoordeling. SHERPA gebruikt als voorbeeld een begrenzend model van de dynamica om de toekomstige toestanden na een reeks acties te voorspellen: hoe sneller de berekening van deze voorspelling, hoe hoger het aantal beoordelingen van acties dat gemaakt kan worden tijdens de online exploratie.

De graaf wordt in drie stappen gegenereerd, voorafgaand aan de exploratie. Eerst wordt een willekeurige subset van de set van mogelijke toestanden geselecteerd als de *operationele set* van de RL-entiteit. Daarna worden de set van toestanden, de set van acties en de tijd gediscretiseerd. Tot slot wordt het begrenzende model uitgevoerd om de zijden te genereren die de knopen van de graaf verbinden. Het resultaat is een hypegraaf die een overschatting is van het begrenzende model. De hypergraaf wordt opgeslagen als een matrix, zodat het online berekenen van de transities vervangen wordt door een indexeringsoperatie, wat minder rekenkracht vereist dan het toepassen van het begrenzend model. Bovendien wordt een gunstige toestandsdiscretisatie geïntroduceerd in de vorm van tegels die op gelijke afstand van elkaar staan. Het resultaat in dat het genereren van de graaf in complexiteit vermindert waardoor het gebruikt van worde voor online exploratie.

De graafformulering is geïmplementeerd binnen de SHERPA strategie door middel van twee graaf-gerelateerde *veiligheidsmetrieken*. Deze kennen aan iedere knoop van de graaf een gewicht toe, welke volgens de gegeven veiligheidsmetriek correspondeert met de veiligheid van die knoop. De veiligheidsbeoordeling wordt hierdoor omgezet in een optimalisatie probleem, zoals ook gebeurt bij de OptiSHERPA strategie. De metrieken zijn geïmplementeerd voor zowel een simpel quadrotorsysteem als voor een vliegtuigbesturingstaak door middel van het hoogteroer. De *operationele* metriek is effectiever in taken waar risico's makkelijker voorspeld kunnen worden, zoals in de quadrotor taak. Voor complexere taken, zoals de hoogteroer taak, is de *nabijheids* metriek effectiever doordat het de evolutie van het systeem over de tijd beperkt.

Als aangenomen wordt dat de operationele set geen fatale toestanden bevat, dan kan de veiligheidsbeoordeling van het gehele beleid uitgevoerd worden door *verwijderen van knopen in de graaf (snoeien)*. Zijden die de operationele set schenden worden verwijderd uit de graaf, samen met de knopen die, als resultaat van het verwijderen van deze zijden, geen uitgaande zijden meer hebben. Alle beleidsstrategiëen die verenigbaar zijn met de gesnoeide graaf zijn daarmee automatisch veilig; echter, het tegenovergestelde is niet waar vanwege de onzekerheid in de graaf. Snoeien kan geschikt gemaakt worden voor online exploratie van operationele sets die langzaam variëren in de tijd, zoals wordt nagebootst in een aantal taken waarin een MAV door een gang vliegt.

Onzekerheid in een model van de omgeving resulteert in de uitdaging van *robu-*

ustheid. Hoe onzekerder de omgeving is, hoe meer het beschikbare model de werkelijke dynamica moet overschatten om nog steeds een begrenzend model te vormen en hoe fijner de resolutie van de discretisatie van de graaf moet zijn om de overschatting van het afgelegde pad te bevatten. In dit proefschrift wordt dit probleem gemitigeerd door implementatie van *Hierarchical Reinforcement Learning* (HRL). Door abstractie van de toestandsruimte, door het toevoegen van ontwerpkennis en door het beperken van de ruimte van ontdekbare beleidsstrategiëen kan HRL bijdragen aan zowel de uitdaging van robuustheid als aan de uitdaging van veiligheid. De innovatieve combinatie van HRL methodes met betrekking tot veiligheid van RL wordt gepresenteerd als Safe Hierarchical Reinforcement Learning (SHRL).

In dit proefschrift wordt *Virtuele Veiligheidstraining* (VST) voorgesteld als SHRL methode en deze bestaat uit drie stappen. Als eerste wordt de originele toestandsruimte getransformeerd, via een willekeurige *projectie functie*, op een manier dat ten minste een geprojecteerde toestandsvariabele onafhankelijk is van en/of relatief is ten opzichte van de andere toestandsvariabelen. Dit maakt het mogelijk om de complexiteit en de onzekerheid van de omgeving te verminderen. Als tweede stap wordt een *belief* set geïntroduceerd, die de mogelijke projectie van de fatale toestandsruimte weergeeft. Daarna wordt in de virtuele geprojecteerde leerruimte een initieel beleid geleerd voor ieder element van de belief set. De doeltreffendheid van deze methode hangt af van de volledigheid van de belief set en van de onzekerheid in het geprojecteerde begrenzende model. Uit experimenten met een MAV die een doel moet vinden in een rommelige omgeving blijkt dat deze strategie veiliger is, zelfs met een simpele belief set en een ongeavanceerd model.

Als tweede en laatste SHRL methode wordt Knoop-Classificatie (VC) geïntroduceerd. Deze methode integreert alle voorgaande bijdrages, zoals veiligheidsoptimalisatie, graafformulering van de dynamica en projectie van de toestandsvariabelen, om zo sequentieel het probleem van veilige exploratie op te lossen. Eerst wordt een operationele set voor de geprojecteerde toestandsruimte gedefinieerd en wordt een graaf gecreëerd van de geprojecteerde dynamica. De zijdes die leiden tot een schending van de operationele set worden geordend op volgorde van het niveau van onwenselijkheid, wat vervolgens gebruikt wordt om voor elke knoop en voor iedere schending twee sets van gewichten te berekenen, de *niveaus* en de *coefficiënten*. Tot slot worden deze gewichten gebruikt om de veiligheid van de zijdes in de graaf te schatten door aan iedere schending een *intensiteit* toe te kennen. Het *veiligste beleid* van VC, gevalideerd in dezelfde twee MAV taken, voorkomt alle botsingen wanneer getest met verschillende modelrealisaties en met verschillende indelingen van obstakels. Bovendien is er een gelijkenis met potentiaalveld methodes, bijvoorbeeld in het vast komen te zitten in lokale minima tussen het doel en de obstakels.

Als een laatste beschouwing betreffende het hoofddoel, het ontwikkelen van veilige, online en robuuste RL exploratiemethodes voor UAVs, draagt dit proefschrift bij aan de huidige stand van de wetenschap door zowel het leveren van meerdere methodes die iedere uitdaging afzonderlijk mitigeren, als door het leveren van hybride algoritmes, die meerdere uitdagingen tegelijkertijd aanpakken. Expliciete en heldere aannames voor toepassing van de voorgestelde methodes zijn gegeven; naast deze aannames zijn de methodes ontwikkeld in overeenstemming met de principes van autonomie, aanpassingsvermogen en modelonafhankelijkheid binnen RL, voor zover het probleem van veilige exploratie dit toestaat. Desondanks worden in dit proefschrift verschillende punten voor verdere ontwikkeling en verbetering aangedragen, gericht tot zowel de *machine learning* als de UAV-operator gemeenschap.

Nomenclature

List of symbols

$\boldsymbol{x}$	State
$\mathcal{S}$	State set
u	Action
$\mathcal{A}$	Action set
$r, \mathcal{R}$	Reward and reward Function
$\mathcal{D}$	Transition function
$\hat{\mathcal{D}}$	Bounding model
γ	discount factor
V, Q	Value and action-value function
π	Policy
σ	State trajectory
$[*]$	Interval of $*$
τ	Transition, interval trajectory, tile
$\mathtt{i}$	Tile index
$\mathcal{G}$	Hypergraph
μ	Projection function
$\boldsymbol{p}^i$	i^{th} projected state

Abbreviations

RL	Reinforcement learning
SRL	Safe reinforcement learning
HRL	Hierarchical reinforcement learning
MDP	Markov decision process
SMDP	Semi-Markov decision process
POMDP	Partial observable Markov decision process
OE	Operational envelope
$\mathrm{SSS}, \mathrm{FSS}$	Safe and fatal state space
$\mathrm{PSSS}_i, \mathrm{PFSS}_i$	Projected safe and fatal state space of $\boldsymbol{p}^i$
TEA	Temporally extended action
MOS	Machine, option or subtask
OM	Operative metric
PM	Proximity metric
AFF	Artificial force field

Acknowledgements

Both me and this dissertation owe considerably to the many people that I had the luck of meeting during my four and a half years of PhD. To all of you goes my most sincere gratitude: whether it was your intention or not, I am the better for having met you. Still, I want to take some space to thank some of you more in detail.

To Dr. ir. Erik-Jan van Kampen, my supervisor and co-promotor, I wish to express my utmost respect and indebtedness. His support during my PhD years has been invaluable, from helping with daily problems in my research to advising me in further career choices. Erik-Jan contributed to make my first months of PhD manageable and untroubled, despite all the adjustments that new research topics, project responsibilities, and different customs required. Erik-Jan, thank you. You not only mentored me in my craft, for which I will be forever indebted: you taught me how to be a better researcher, a better teacher and a better thinker as well.

To Prof. dr. ir. Max Mulder, my promotor, goes my highest esteem and my deepest gratitude for giving me the opportunity of pursuing a PhD. Prof. Mulder was there through some of the most difficult parts of my PhD, especially toward its end, and helped me in many ways. Professor, without your great personality, trustworthy opinion, proactive attitude, and efficient feedback, I don't think this dissertation would have seen the light of day as early as it did. Thank you.

I want to thank Dr. Qiping Chu and Dr. ir. Coen de Visser for their sound and helpful advices during my research and during the writing of this dissertation, as well as for the conversations on the most diverse topics that we enjoyed during the past few years. Dr. ir. René van Paassen made our brief collaboration in my last few months of Phd very enjoyable, and for this I wish to thank him. I am really thankful to Bertine Markus for her valuable support in the convoluted coils of bureaucracy. Moreover, I am grateful to everyone in the staff for making C&S a pleasant and accommodating place to be. I thank all the students I had the pleasure to collaborate, supervise, or interact with, for their hard work, high spirits, and good attitude.

My PhD colleagues at C&S have played a pivotal role in keeping me relatively sane during these past years. My "madrigal companions", Jaime, Sophie, and Tao, have a lasting place in some of my fondest memories. Thank you so much for all the time we spent together. João, Dyah, "Czech" Jan, Hann Woei, Emmanuel, Peng, Annemarie, Ye, Shuo, and Sihao are part of the very special Simona 4.04 group. Thank you all, especially my desk-mates João and Annemarie, for your pleasant company and for making the room such a good place to work in. Thank you to my "skiing colleagues" Jaime, Sophie, Matěj, Kim, and Isabel, who put up with me on the piste and who made me discover a new passion. Thank you to my seniors, "tall" Jan, Maarten, Rolf, Yazdi, Deniz, Liguo, Rita, Laurens, Jia, Mariam, Paul, Herman, for being an inspiration to me, for their friendliness and modesty, and for their savvy advices. Thank you to the rest of my PhD friends, Julia, Kirk, Sherry, Gustavo, Yingzhi, Ewoud, Diana, Sarah, Neno, Jerom, Ivan, Junzi, Dirk, Yke, Wei, Kasper, Sjoerd, Mario, Ye, Henry, and Jelmer. I am really thankful to all of you and I treasure the time we spent together.

Many other people outside of the department contributed to my well-being and thus to this dissertation. Jay, Ranko, Arturo, Dena, Alexey, Encar, Daan and many others shared their love of music with me, as well as their precious company, putting my otherwise troubled mind to *requiem*. Eftychia, Bas, David, Jelle, Rossiza, Mihai, Matei, Panos, thanks for being often *on-board* with such a peculiar *character* as me. I want to thank Iris for her friendliness, as well as for her insightful advices on both cuisine and work, and Martina for sharing her taste in movies. When I look back, I feel so lucky to have met every single one of you, and I hope we'll stay in contact in the future.

To my Italy-bound friends goes a huge, collective *grazie*. Giulio, Lorenzo, Federico "Ciappa", and Giulia, thank you for always being in reach. You are family to me, and I will always have a spot for you in my heart. Luca, Lorenzo "Sasso", Luca "Bimbo", Andrea, Luca "Dello", Federico "Chico", thank you for sharing your precious company with me, whether during my holidays back home, or when visiting. Everyday I felt as if I had just left. A very special thank you goes to Filippo, my cousin, who I love as a brother. To Federica, so talented and friendly, goes a big, big hug. Thanks to you all, South Holland and Tuscany have been so much closer to each other.

To my father, my mother, my sister, all my family, goes most of the credit for whatever good and decent is in me or comes from me. Thank you for being who you are, from the bottom of my heart. Thank you for having been there for me, always. Thank you for your unceasing support and affection, for your efforts, for having faith in me. You made all this possible. I love you.

Alas, I am indebted to too many of you for all to be mentioned. I wish you all the best. I won't forget any of you.

Delft, 12 October 2017

Tommaso Mannucci

Curriculum Vitæ

Tommaso MANNUCCI

24-10-1987	Born in Pisa, Italy.

Education

2006–2009	BSc, Aerospace Engineering University of Pisa, Pisa, Italy
2009-2012	MSc, Aerospace Engineering University of Pisa, Pisa, Italy
2013-2017	PhD., Aerospace Engineering Delft University of Technology, Delft, the Netherlands *Thesis:* Safe Online Robust Exploration for Reinforcement Learning Control of Unmanned Aerial Vehicles *Promotor:* Prof. dr. ir. M. Mulder

Professional activities

2013	Transmission Test-bench Designer AM Testing srl Pisa, Italy
2013-2014	Camera Payload Specialist Delft University of Technology - 2seas-3i project

Academic activities

Reviewer for the Journal of Guidance, Control and Dynamics

Reviewer for the IEEE Transactions on Neural Networks and Learning Systems

Reviewer for the International Conference on Intelligent Robots and Systems (IROS)

Reviewer for the Annual Conference on Neural Information Processing Systems (NIPS)

List of Publications

9. **T. Mannucci**, E. van Kampen, C. C. de Visser and Q. Chu, *Safe Exploration Algorithms for Reinforcement Learning Controllers*, in IEEE Transactions on Neural Networks and Learning Systems (accepted for publication).

8. **T. Mannucci**, E. van Kampen, C. C. de Visser and Q. Chu, *Safe and Autonomous UAV Navigation using Graph Policies*, in AIAA Information Systems-AIAA Infotech, AIAA SciTech Forum 2017, Grapevine, TX.

7. **T. Mannucci**, E. van Kampen, C. C. de Visser and Q. Chu, *Hierarchically Structured Controllers for Safe UAV Reinforcement Learning Applications*, in AIAA Information Systems-AIAA Infotech, AIAA SciTech Forum 2017, Grapevine, TX.

6. **T. Mannucci** and E. van Kampen, *A hierarchical maze navigation algorithm with Reinforcement Learning and mapping*, in 2016 IEEE Symposium Series on Computational Intelligence (SSCI), Athens, Greece.

5. S. Shyamsundar, **T. Mannucci** and E. van Kampen, *Reinforcement learning based algorithm with Safety Handling and Risk Perception*, in 2016 IEEE Symposium Series on Computational Intelligence (SSCI), Athens, Greece.

4. **T. Mannucci**, E. van Kampen, C. C. de Visser and Q. Chu, *Graph based dynamic policy for UAV navigation*, in AIAA Guidance, Navigation, and Control Conference, AIAA SciTech Forum 2016, San Diego, CA.

3. **T. Mannucci**, E. van Kampen, C. C. de Visser and Q. Chu, *A novel approach with safety metrics for real-time exploration of uncertain environments*, in AIAA Guidance, Navigation, and Control Conference, AIAA SciTech Forum 2016, San Diego, CA.

2. J. Junell, **T. Mannucci**, Y. Zhou and E. van Kampen, *Self-tuning Gains of a Quadrotor using a Simple Model for Policy Gradient Reinforcement Learning*, in AIAA Guidance, Navigation, and Control Conference, AIAA SciTech Forum 2016, San Diego, CA.

1. **T. Mannucci**, E. van Kampen, C. C. de Visser and Q. Chu, *SHERPA: a safe exploration algorithm for Reinforcement Learning controllers*, in AIAA Guidance, Navigation, and Control Conference, AIAA SciTech Forum 2015, Kissimmee, FL.