Safe Reinforcement Learning: Applications

Tiago Nunes

TU Delft

Safe Reinforcement Learning: Applications

by

Tiago Nunes

to obtain the degree of Master of Science
at the Delft University of Technology,

Student number:	4757122	
Project duration:	February 1, 2019 – December 11, 2019	
Thesis committee:	Prof. dr. ir. Erik-Jan van Kampen,	TU Delft, supervisor
	Prof. dr. ir. Q.P Chu,	TU Delft, Chairman
	Prof. dr. ir. W. van Der Wal,	TU Delft, External examiner
	B. Sun, MsC	TU Delft

TU Delft

"The fact that we live at the bottom of a deep gravity well,
on the surface of a gas covered planet
going around a nuclear fireball 90 million miles away and think this to be normal
is obviously some indication of how skewed our perspective tends to be."
-Douglas Adams

List of Abbreviations

ACD Actor Critic Designs .

ADHDP Action Dependent Heuristic Dynamic Programming.

ADP Adaptive Dynamic Programming.

GP Gaussian Process.

HDP Heuristic Dynamic Programming.

ICAO International Civil Aviation Organization.

LPV Linear Parameter-Varying (controller).

LQR Linear Quadratic Regulator.

MAV Micro Aerial Vehicle.

MDP Markov Decision Process.

MIMO Multiple Input Multiple Output (system).

ML Machine Learning.

MOMDP Multiple Objective Markov Decision Process.

NAC Natural Actor Critic.

NN Neural Network.

PID Proportional, Derivative, Integrative (Controller).

POMDP Partially Observable Markov Decision Process.

RL Reinforcement Learning .

ROA Region Of Attraction.

SafeRL Safe Reinforcement Learning.

SMDP Semi-Markov Decision Process.

TD Temporal Difference, a type of RL method.

UAV Unmanned Aerial Vehicle.

VIQL Value Iteration Q-Learning.

List of Figures

List of Tables

Contents

1 Introduction

1.1. Motivation for the thesis

Over the last years unmanned aerial vehicle (UAVs) have become more and more prolific both in civilian and military markets. One of the reasons for this widespread proliferation and usage has to do with the adaptability and versatility of UAVs and their relative low cost (when compared to conventional aircraft). UAVs are currently used in several different domains such as: farming, surveillance, photography, conservation and law enforcement.

One type of UAV that has become more and more common among the civilian marked is the Micro Aerial Vehicle (MAV), due to it's small size, cost and relative ease of operation.

Due to the fact that these UAVs are becoming increasingly common it is important to come up with controllers for these aircraft that assure that they fly safely and efficiently, even more so for drones that are meant for industrial applications where relatively small errors can generate a significant loss.

The dynamics associated with UAVs are usually highly non-linear and "fast" which means that finding a model that accurately describes their behavior can be hard. This presents a problem when designing a controller for a UAV, if a model is not available or has a high level of uncertainty then the performance of the controller might be compromised.

When designing a controller one can choose several different options (as shown in figure 1.1). For simpler systems PID controllers might be enough to accomplish the task at hand. For more complex systems, however, when there is uncertainty to the model provided or in case a failure or sudden change of the model happens a more complex approach is required, such as "Fault-Tolerant Controllers" that can be either model based (if they use information about the model) or model free (if they don't rely on any explicit representation of the model). Another way of distinguishing different controllers is for their linearity, there are controllers adequate for linear systems (such as conventional PIDs) and those designed for non linear systems (such as PIDs with gain scheduling), the focus of the work described in this report, however, will be focused on the difference between static and fault tolerant controllers and not on the linear/non-linear dimension.

For tasks where a model of the system is hard to obtain or not obtainable a Model Free approach is usually more suitable. Reinforcement Learning is one of the methods one can use to design such controllers, it uses an iterative process where, through continuous exploration of the state space, the agent will learn which are the best actions to perform in each state. The agent is given, at each time step, a reward that depends upon how desirable the action taken is.

The system is initialized in a given condition, the agent will then iteratively improve its value function estimates and will seek to maximize the reward received through experimentation, turning a control design problem into an optimization one. Reinforcement Learning is a bio-inspired learning process that aim to mimic animal behavior, certain actions are attempted and a reward is given, based on that reward the agent will choose what's the best possible action in future situations.

The only thing the designer of the controller needs to provide the system (agent) with is a reward at each time step. For certain applications designing a reward function can be extremely simple, for example, in programming a robot to exit a maze the reward can be simply defined as a positive value (like +1) when the robot exits the maze and a negative value (like -1) for each time step the robot spends in the maze, this encourages the robot not only to leave the maze but also to do it as fast as possible. Therefore this principle

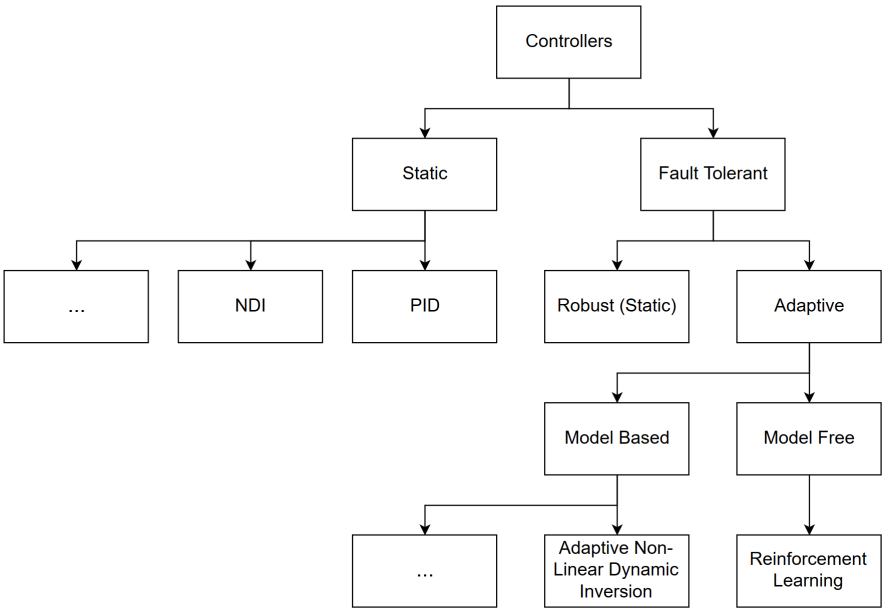


Figure 1.1: Different types of controllers

can also be applied when designing a controller be it for a system with highly non-linear dynamics such as a UAV or for a more conventional fixed wing aircraft such as a Boeing 787 Dreamliner.

If the reward is well designed then by maximizing the expected sum of future rewards the agent will be learning an optimal control policy. There are numerous different RL algorithms to achieve this.

1.2. Reinforcement Learning: An Introduction

Reinforcement Learning can be described as an iterative process:

1. The agent is initialized at state S_t (t=0), with a random value function
2. The agent takes action A_t and transitions to state S_{t+1} receiving reward R_t
3. The agent updates the value function to reflect the new information
4. Repeat from 2 until some performance measurement is achieved or until a number of episodes have been concluded

The benefit of using Reinforcement Learning is that a policy can be obtained without giving the agent any information as to the model of the vehicle or process that is to be controlled. Another benefit is that Reinforcement Learning can be done both off-line and online.

- **Offline Learning** consists of learning from past experienced/simulated data.
- **Online Learning** consists of learning through experimentation.

Most RL methods are developed to solve problems that can be classified as **Markov Decision processes (MDPs)** [Barnard, 1993]. If a problem is an MDP then if the system is at state *s* and executes action *a* then the state *s'* and the reward *r* it receives are only dependent upon *s* and *a*, meaning it does not depend on previous actions and states. There are also RL methods that have been extended to work in Partially Observable MDPs (POMDPs)([Lovejoy, 1991]) and with Semi-MDPs(SMDPs, [Sutton et al., 1999]). POMDPs are problems where the underlying process is an MDP but the agent is unable to determine (observe) the current state. SMDPs are MDPs where the time between two consecutive actions is a random variable.

The problem with using RL is that there is a need to explore the state space of the problem in order to find an optimal policy but, at the same time, there is also a desire for the agent to act optimally in order to maximize the reward, this is the so called **Exploration VS Exploitation dilemma**. Initially the agent has to take mostly random actions, even if they are not (to the agent's current knowledge) the best possible actions,

in order to find other states and actions that might be more rewarding then the ones it already experienced. As the learning progresses the agent already starts having a good enough idea of the state space and has probably converged to a policy that is close to the optimal one so it would be desirable for the agent to exploit the best possible actions instead of taking random actions in order to maximize the reward obtained. There are numerous methods that attempt to deal with this dilemma, one of the most popular ones is the so called ϵ_{greedy} method [Pearl, 1984].

$$A_{t+1} \begin{cases} \text{chosen from } \pi, \text{ with probability } (1-\epsilon) \\ \text{random, with probability } \epsilon \end{cases} \tag{1.1}$$

Where π is the current policy. ϵ can be initialized with a high value, for example 0.2, and then be progressively reduced to allow the agent to eventually start using the greedy policy more and more.

The question then is: if RL can be used to learn a near-optimal control strategy for a system by simply designing a reward function that rewards the desirable behavior promote, why is it not used for every system?

The answer lies in the **Exploration VS Exploitation dilemma**, in order to guarantee that policy that the agent learns converges to the optimal policy, each state needs to be visited an infinite (or, in practical terms a very large) number of times. For systems that don't have any unsafe states this is not a problem since, in this case, the agent can just be allowed to learn for a long time. The problem arises because most systems, and UAVs in particular, have failure states. UAVs are low-flying so they are liable to hit other objects or even people, they can also be damaged themselves due to collisions or falls. This means that a controller that just takes random actions initially can lead to unsafe or failure states that can cause damages.

There is, therefore, a need to find a way to achieve exploration of the state space such that the optimal control strategy can converge to its optimal value without compromising safety and leading to unsafe states due to random action taking. This is where Safe Exploration Reinforcement Learning (SafeRL) methods come in. SafeRL methods are designed in such a way that the agent explores the state space in order to find the optimal policy but also avoids taking risky actions. SafeRL methods' two goals are maximizing expected rewards and satisfying safety constraints.

This risky action avoidance can take many forms: from a function that defines a go/no-go decision for each move ([Hans et al., 2008], [Pecka et al., 2015]), to algorithms that give the agent a good initial demonstration that it tries to improve on ([Driessens and Džeroski, 2004], [Argall et al., 2009],[Martínez et al., 2015]), to worst case or *minimax* methods that take safety as an optimization problem (if low reward is given to unsafe states then the agent will learn not to go to those states) ([Heger, 1994],[García and Fernández, 2015].

Most approaches in the RL field that try to guarantee safety do it in one of two ways:

- Integrating safety into the optimization of the cost function, which is usually done by giving a very negative reward for a transition that is deemed unsafe or that possibly leads to an unsafe state
- Constraining the explorable safe space, which works by not allowing the agent to transition to state that are deemed unsafe or uncertain.

Although both ideas can be effective, for the purpose of this research the latter one will be focused on the most. The reasoning behind this is because of some characteristics when applying the former approach, by including safety into the cost function unsafe transitions are being punished harshly but that doesn't mean they won't still occur. Occasional occurrence of critical or unsafe transitions might be acceptable for certain applications, for example a waiter robot dropping a few plates now and then, but it certainly is not acceptable when it comes to the UAV or commercial aircraft domain or even in the domain of self driving cars since it can lead to loss of vehicle or loss of life (in the case of commercial aviation, catastrophically so). Therefore, for these kinds of applications, where safety is **critical**, it is preferable to use methods that constrain the explorable safe space, even if this might lead to a suboptimal policy and certain possibly-optimal regions of the state action space not being explored. As is commonly stated "taking-off is optional, landing is mandatory".

It is important to define what exactly is understood by the word **"safety"**. One could use, for example, data to back up the fact that air travel is much safer than cars by looking at the number of casualties per km traveled/flown. As stated in ICAO's Doc 9735, **"Safety Oversight Manual"** [Organization, 2011], safety is defined as:

"A condition in which the risk of harm and damage is limited to an acceptable level."

What corresponds to an acceptable risk is, of course, determined by the situation itself. So an acceptable risk when flying a commercial plane would have to correspond to a much lower chance of failure than, for example, the acceptable risk of a toy car malfunctioning.

For the basis of this research a similar definition will be applied, an algorithm will be described as one executing **Safe**RL when it avoids transitioning to critical or unsafe states the vast majority of times.

Another problem that often has to be addressed when it comes to RL has to do with computational efficiency of training. This is even more critical when online training is concerned, the training has to be efficient enough for it to be done in a reduced amount of time. This is due to the fact that during online training the agent is learning and acting at the same time, so it has to be able to decide what the next action will be fast.

1.3. Research Goal

The main goal of the research carried out in this report is to investigate what the current state of the art in Safe Reinforcement Learning is in order to construct an algorithm that would allow an agent to safely explore a given state space and eventually converge to a near-optimal policy.

1.4. Research Questions

The research questions that will guide this dissertation are as follows:

1. How can Reinforcement Learning be used in control tasks for safety-critical systems?
2. What is the current state of the art in Safe Reinforcement Learning
3. Of the two main approaches to Safe Reinforcement Learning which one is more adequate?
4. How can an algorithm be designed that achieves Safe Reinforcement Learning?
    - How can the operative and proximity metrics be combined in order to achieve a SafeRL algorithm?
    - How can the ellipsoids discussed be used as a way to reduce model uncertainty?

1.5. Report Structure

The remaining report will have the following structure: chapter 2 is made up of the scientific paper written about the research conducted in this thesis. Chapter 3 presents an introduction to the concepts behind Reinforcement Learning and some of the algorithms used in it, it will also present the state of the art research conducted about both RL in the flight control and Safe Reinforcement Learning. Chapter 4 is made up of the preliminary results obtained during the Literature Review stage of the thesis. Chapter 5 shows several additional results that were not relevant enough to be displayed directly in the scientific paper. The final chapter of the thesis is chapter 6 where conclusions about the work done will be drawn, the research questions answered and some recommendations for future work presented.

2

Scientific Paper

Safe Reinforcement Learning Applications

Tiago Nunes*
Supervisor: Dr.ir. E. van Kampen†

Reinforcement Learning (RL) focuses on maximizing the returns (discounted rewards) throughout the episodes, one of the main challenges when using it is that it is inadequate for safety-critical tasks due to the possibility of transitioning into critical states while exploring. Safe Reinforcement Learning (SafeRL) is a subset of RL that focuses on achieving safe exploration during the learning process and, thus, allowing it to be used in safety critical tasks. This research focuses on expanding already existing SafeRL algorithms through a combination of two previously developed safety metrics into a novel one. Furthermore, this research also uses an ellipsoid-based bounding model to replace the interval analysis bounding model. To validate this combination of the metrics, two different examples are used to test the performance of the various algorithms and compare their relative survivability and computational efficiency: a quadrotor navigation task and an elevator control task. Results show that the novel combined metric (ProxOp) outperforms one of the metrics in the quadrotor navigation task and the other one in the elevator control task. Overall, the combined metric is a better option for usage when there is no *a priori* knowledge on how any of the metrics will behave. The ellipsoidal bounding model is tested using the elevator control task and is shown to have comparable performance when used in combination with the proximity and the ProxOp metrics but shows a significant degradation of performance when used solely with the operative metric. The ellipsoidal bounding model is also preferable for use in tasks where there is no knowledge on what are the bounds of the system model, as the ellipsoidal bounding model estimates the model error initially through the use of Gaussian processes. This research, thus, presents a viable novel safety metric as well as an alternative bounding model that can be used with it for applications of RL where safety is important.

I. Introduction

Recent decades have seen an increasing rate of development of both Unmanned Aerial Vehicles (UAVs) and Machine Learning-based control systems. These two different areas lend themselves to cooperation since it can be expensive to get an adequate mathematical model to describe the behavior of the UAV and RL is a Machine Learning algorithm that can be done model free.

Unmaned Aerial Vehicles (UAVs) have become more prolific in both civilian and military applications, such as farming, surveillance, photography and law enforcement it is, thus, important to come up with controllers that are adequate and make sure they fly efficiently and safely. UAVs are characterized by highly non-linear and "fast" dynamics which are hard to describe mathematically. Therefore, it is preferable to use a controller that is model-free instead of model-dependant. A controller that is to be used by a UAV also needs to be able to adapt to changing conditions of the environment. When it comes to model-free adaptive controllers, one of the methods used is called Reinforcement Learning (RL), a subset of Machine Learning algorithms.

RL works on the principle that the agent's (UAV in this case) goal is to maximize the reward signal received when interacting with the environment [Sutton and Barto, 2017]. One of the advantages of RL is that it can be done fully model free, the designer need only create an adequate reward function. For example, for altitude hold control an adequate reward function would be a function that would give a higher penalty the further away from the target altitude the agent is. RL works by exploring the state space in order to find a good policy for the control task, there is then a balance between continually improving the policy and exploiting the current policy to get a higher reward, the so called *exploration vs exploitation dilemma*.

One of the issues with using RL in mainstream control tasks, however, is that for safety critical domains (such as aviation, where safety is much more important than efficiency) there are states that prove to be fatal to the system (fatal

*MSc. Student, Control and Simulation, Aerospace Engineering, Delft University of Technology (email: T.M.Monteironunes@student.tudelft.nl).
†Assistant Professor, Control and Simulation, Aerospace Engineering, Delft University of Technology.

states). In addition, there are also states that will always lead to fatal states (lead-to-fatal states). When controlling a safety critical system the agent must stay clear of both fatal and lead-to-fatal states. This becomes an issue when it is considered that RL relies on exploring the state-space without, in many cases, any *a priori* information about the environment which means that it is possible that, while exploring, the agent may end up transitioning to a lead-to-fatal or fatal state, which would be a critical occurrence.

It is that issue that this research aims to solve, in order to allow the use of RL algorithms for control of both UAVs and conventional aircraft a way to incorporate safety into these algorithms must be found. The methods that seek to add safety to RL are called Safe Reinforcement Learning (SafeRL) methods. These methods work by one of two main principles ([García and Fernández, 2015] presents a comprehensive study of these two methods and their sub-methods)

The first principle is the **Optimization Criterion** which works by adding safety as part of the objective function, usually involving giving the agent a severe penalty for allowing critical transitions to occur. [Horie et al., 2019]

The second principle is the **State-space Constraining** that works by preventing the agent from taking actions that lead to fatal states by restricting the areas of the state space the agent is allowed to reach. [Martínez et al., 2015] [Li et al., 2018] [Isele et al., 2018] [Berkenkamp et al., 2016] [Luo et al., 2018]

The algorithms presented in this article follow the second principle, the reason for that is that giving a severe penalty when a critical transition occurs might not make that transition as unappealing, due to averaging, as one might first think whereas using state-space constraint can block the agent from taking that transition altogether ([Isele et al., 2018]). In domains where safety is paramount the interest is in stopping critical transitions, which means that in these domains the second principle is more adequate. Since this research focuses on aviation applications, the State-Space Constraint principle will be used.

The contributions of the work presented in this article are twofold: firstly the two safety metrics previously developed in [Mannucci, 2017] are combined into a novel combined metric called the **ProxOp** metric, secondly the interval analysis bounding model used is replaced with an ellipsoidal based bounding model [Torsten Koller and Krause, 2018]. The combined metric is tested for two different tasks, quad-rotor navigation and elevator control, and the ellipsoid model is tested for all three metrics in an elevator control task. Results show that the **ProxOp** metric is a suitable metric and that, when no information about which metric will perform better is known *a priori*, it is a better choice. Results also show that the ellipsoidal bounding model is viable although it does show some performance degradation, especially when used in combination with the operator metric. Ultimately the ellipsoidal bounding model would be preferable when the system model error is unknown *a priori* since the initial steps of the process involve estimating it whereas the previous interval analysis bounding model needed *a priori* information on the bounds of the system model.

The structure of the article will now be introduced. Firstly in section II a brief introduction to the RL methods used are presented where the algorithms and principles they are based on are explained. In section III the way ellipsoids can be used as a bounding model for multi-step-ahead predictions of state transitions are explained. Section IV presents the adaptations needed for the ellipsoidal bounding model to work with both metrics. In section V two different tasks are presented alongside their models and all the parameters used during simulation, following this, in section VI, the results of the simulations are presented and discussed. Finally, section VII presents some conclusions regarding the research developed and discuss future work that can be done to expand upon it.

II. RL: Introduction to the methods used

RL works by looking for the optimal policy, i.e the policy that maximizes the reward function for the given problem. A policy is merely a mapping that tells the agent which action to take in each state. In order to find this optimal policy the agent is initialized with a given policy and value function (which gives the expected return of a given state) and will update both iteratively, i.e the agent will use the initial policy to experience the actual rewards of given state transitions and thus update the value function which will, then, change the policy. To calculate the value of a state (or state-action pair) **returns** are used, **returns** take into account not just the reward given by the state transition but also the expected value of all further transitions (each one discounted by a factor that is smaller the further time steps away it is from the current state).

The methods that will be used are based in the same logic and methodology of looking for an optimal policy through exploration and maximizing returns but they differ from most conventional RL methods in that they focus on safety of exploration, even though that may lead to a sub-optimal policy being achieved. This means that the algorithms developed have to have a way to differentiate the safe states from the fatal states which can be done, for example, by using a sensor (such as a distance sensor to avoid collision with an object or wall). Ultimately the goal of a SafeRL algorithm is to explore a state space and find a near-optimal policy while, simultaneously, managing to stay within the

safe state space of the system.

A. Safety Metrics

The Safety Metrics methods are hybrid methods that combine the principles of the SHERPA/OptiSHERPA algorithms with Graph Generation methods and a safety metric to achieve safe exploration [Mannucci, 2017].

The basic idea behind them is that Graph Generation is used to generate the possible state transitions (a pre-computing task) and thus save time during real-time computation, the safety metric is then used in combination with these transition graphics to decide which decision to be taken. The way the metrics are used can be seen in figure 1, the bounding models are used to calculate the n-step ahead prediction of the state when a certain action is used. An episode ends when either the maximum number of iterations has been reached or a end of episode event happens (such as a crash or a failure).

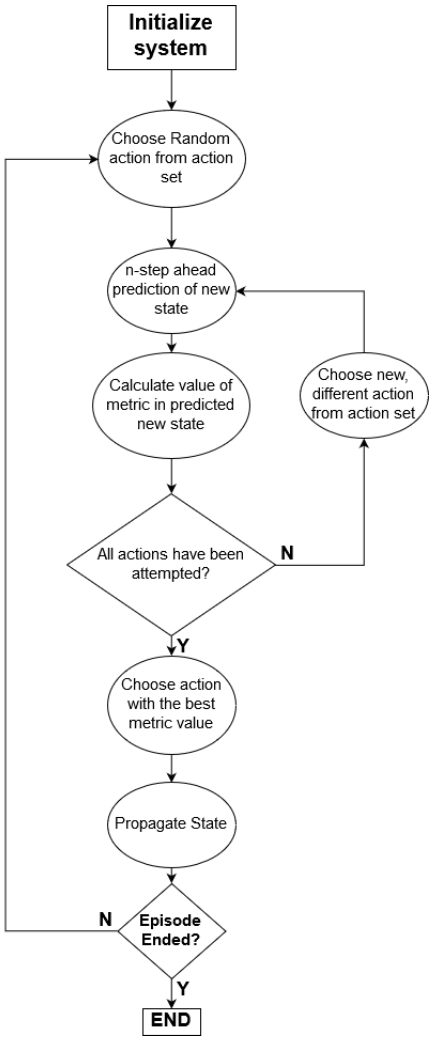


Figure 1 Flow diagram for the metric functioning.

In [Mannucci, 2017] two metrics were introduced:

- **Proximity Metric:** accounts for the distance of the system to the closest previously-visited point of the state space.
- **Operative Metric:** uses a warning function to continuously update the information it has on the state space, updating which parts of it are part of the Safe State Space or the Fatal State Space accordingly.

These metrics will be further explained in the following subsections.

During the operation of the algorithm, therefore, the system only needs to choose the action at each time step that maximizes the given metric's value.

When using graph generation methods, in general, and these metrics, in particular, for continuous-time tasks then the state space of the system has to be discretized. One such way to discretize the state space is by using "tilings", where a continuous state space is replaced by a series of tiles that are defined by dividing each dimension of the state space into k_i tiles of equal size. Considering the state space is n-dimensional, the total amount of tiles will be equal to $\prod_{i=1}^{n} k_i$.

Tiles are used by the metrics to keep track of the previously visited states (in the case of the proximity metric) or to keep track of the current 'value' of each part of the state space (in the case of the operative metric). An example of how tiles can be used to discretize a state space is shown in figure 2, here a 1×2 meter room is discretized using tiles. For the x_1 direction a tile width of 0.25m was chosen whereas for the x_2 direction a tile width of 0.50m was used, this leads to the system being discretized into a 4×4 grid of tiles whose indexes are shown in the figure as well. The tile indexes can then be used to describe in which tile the agent is currently at, if it is exploring the room for example.

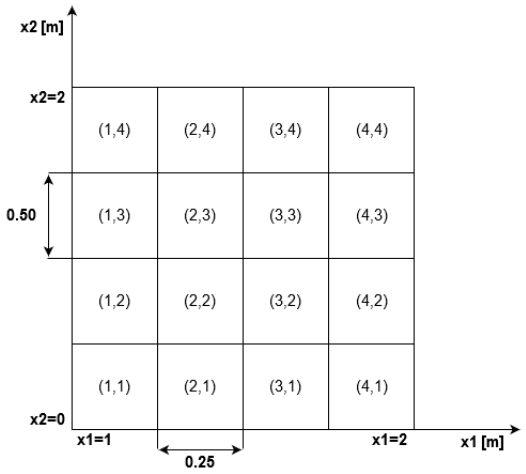


Figure 2 Example of tilling applied to a 2-dimensional state space.

The metrics are defined in such a way that the candidate commands to be tested are of the form:

$$\mathbf{a} = (u(t), u(t+\Delta t), \cdots, u(t+(k-1)\Delta t)); \qquad k_{min} \leq k \leq k_{max}; \quad 1 \leq k_{min}; \tag{1}$$

$$u(t) = u(t+\Delta t) = \cdots = u(t+(k-1)\Delta t); \tag{2}$$

where k is defined as the length of the command. Equation (2) specifies that the commands to be considered are constant commands.

Each of these two metrics will now be described in detail.

1. Proximity metric

The proximity metric focuses on the distance of a given point to the closest, previously visited, point in the state space. Throughout the length of the episode the agent will put all the tiles of the visited states into a list (τ_{list}). Since the tiles are equally spaced on each dimension, the distance can be defined in terms of the distance in indexes of the tiles they belong to.

The proximity of a given point, τ, is defined as:

$$prox(\tau) = -\min_{\tau' \in \tau_{list}} dist(\tau, \tau') \tag{3}$$

Since the indexes of the tiles τ and τ' can be respectively described as: $i(\tau) = (i_1, i_2, ..., i_n)$ and $i(\tau') = (i'_1, i'_2, ..., i'_n)$ (where i_k, $k \in \{1, .., n\}$ is the index of the tile corresponding to the k-th state), the distance can be defined as:

$$dist(\tau, \tau') = ||\mathbf{v_r} \odot (i(\tau) - i(\tau'))||_2 \tag{4}$$

where $\mathbf{v_r}$ is a rescaling vector, $\odot$ defines the Hadamard product of two matrices and $|| * ||_2$ is the Euclidean norm of *.

The rescaling vector is defined as a vector of positive gains that weighs which components of the state space are more or less relevant depending on whether their rescaling vector value is higher or lower, respectively.

Due to the uncertainty of the system, the predicted arrival set of the system after a given set of actions are performed can be a combination of tiles, it is also important to define the proximity for a collection of tiles, C:

$$prox(C) = prox(\tau_C) - \rho \max_{\tau' \in C}(\tau_C, \tau') \tag{5}$$

where τ_C is the center of collection C. The first term accounts for the distance from the collection to the previously visited states and the second term accounts for the dispersion of collection C. As mentioned above, the best action when using the proximity metric is simply the one that maximizes the value of the proximity value. The predicted arrival set for a candidate command is calculated and the proximity of that arrival set is calculated, this action is repeated for the different candidate commands and then the best one is chosen to be carried out.

An example of the way the proximity metric works with a collection is shown in figure 3. The gray tiles around τ_C correspond to the collection (C). When using the weight vector of $\mathbf{v_r} = (2, 1)$ the closest previously visited tile is τ', even though τ'' is closer when considering only difference in indexes. Since τ_f is the furthest point from τ_C that belongs to the collection when the weight vector $\mathbf{v_r} = (2, 1)$ is used, this is the tile that corresponds to the second term in equation (5) used to account for dispersion within a collection.

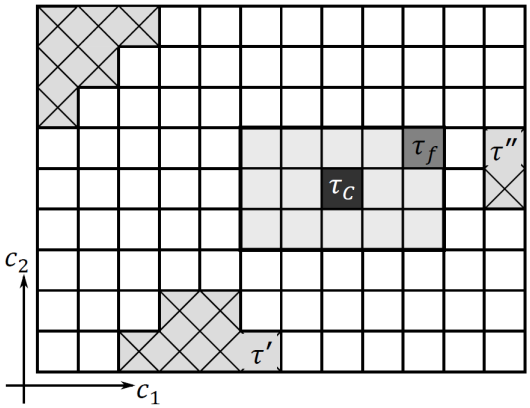


Figure 3 Calculating the proximity of a collection, τ_C is the center of the collection and the tiles with crosses on them are previously visited states. With a weight vector of $\mathbf{v_r} = (2, 1)$ the nearest tile to τ_C is τ'. Image taken from [Mannucci, 2017]

2. Operative metric

The operative metric works by classifying each tile of the state space with a given value based on its safety or lack thereof. The metric works by considering four different possible values for all the tiles in the state space:

- q_{exp}; given to previously visited states.
- q_{safe}; given to states that were determined to be safe but haven't been visited yet.
- q_{unc}; given to states that have not been observed yet.
- q_{fat}; given to states that have been observed and found to be fatal.

where $q_{exp} > q_{safe} > q_{unc} > q_{fat}$

To perform this classification the agent has to have access to a warning function, W, that will indicate whether there is danger close by (W=1) or if the neighbourhood is safe (W=0). This danger function could be performed by a variety of sensors, for example a speedometer (if a velocity limiting task is the one to be performed) or a ranging sensor.

Initially, all tiles are set to have the value q_{unc} to reflect the fact exploration has not happened yet. Throughout the episode, the agent will classify the states it visits by changing their values to q_{exp}. At each time step, if no danger is sensed ($W = 0$) all the tiles in sensing range whose values are not currently q_{fat} will be classified with the value q_{safe}. If, at a given time step, danger is detected ($W = 1$) then all tiles that are within sensing range and whose value is not higher than q_{unc} (unexplored tiles) will have their values changed to q_{fat}.

This algorithm leads to an overestimation of the fatal state space, which is sub-optimal in terms of performance but as the main concern of the algorithm is safety this is acceptable.

When considering a candidate command the algorithm will calculate the arrival set and calculate its value, this process is repeated for all the possible candidate commands and, at the end, as was the case with the proximity metric the candidate command with the highest value is chosen to be carried out. In case the arrival set includes multiple tiles, i.e a collection of tiles (C), its value can be calculated by averaging all the tiles it contains:

$$q(C) = \frac{\sum_{\tau \in C} q(\tau)}{|C|} \tag{6}$$

where $|C|$ is the number of tiles contained in C.

3. Combined metric

One of the contributions of the current research is to seek a way to combine the two metrics described above. The main goal is to achieve a combined metric that will have the best characteristics of the two different metrics. Namely, the operative metric is effective at exploration whereas the proximity metrics is effective at remaining within a given neighbourhood, following a reference signal for example. This combined metric shall be called ProxOp.

There are different ways in which these metrics can be combined. One such way is to simply execute a switch in which metric is currently "in control" of the agent using a given parameter (for example, giving control of the agent to the operative metric only when the warning function outputs 1). Another way of combining the metrics is by following a "membership function" logic and taking into account the choices of both metrics and averaging them by giving each decision a weight. The way the metrics are combined will thus greatly affect performance.

The combination chosen involves using the warning function parameter as a decision variable for switching between the two metrics:

- W=0: meant the system wasn't in danger and thus the proximity metric was used.
- W=1: meant the system was in danger and thus the operative metric was used.

The combined metric then uses the methods described in sections II.A.1 and II.A.2 to calculate the values of the respective metrics and, depending on the value of the variable W, conduct the action that maximized the metric currently in control of the system.

Of course the opposite approach (where the proximity metric is used to respond to danger) is another valid solution, but during testing it was found to have worse performance and thus the operative metric was used to respond to danger throughout this research.

The approach discussed above that involved "membership function" logic uses input from both metrics to decide which action to be used. The action selected at each time step $\mathbf{a}$ would then be chosen based on an equation of the form:

$$\mathbf{a_t} = K_1 a_{op.t} + K_2 a_{prox.t} \tag{7}$$

$$K_1, K_2 \in]0; 1[\tag{8}$$

where $a_{op.t}$ and $a_{prox.t}$ are the actions selected at time t by the operative and proximity metrics, respectively.

When considering discrete sets of actions ($A \in \{a_1, a_2, a_3, ..., a_n\}$) the selected action would then be the closest action to $\mathbf{a_t}$ out of the set.

III. Ellipsoids and bounding models

Due to uncertainties in the system model, it is necessary to have a bounding model while making multi-step ahead predictions of the state. The work done in [Mannucci, 2017] focused on using an interval bounding model, in this research an alternative bounding model is considered, an ellipsoid. Since ellipsoids have appealing geometric properties they have been used frequently by robust control practitioners for this task [Filippova, 2018][Asselborn et al., 2013][Torsten Koller and Krause, 2018].

Ellipsoids provide an over-approximation of the system dynamics, which is desirable when it comes to safe exploration. The over-approximation can then be used to predict whether a given action or set of actions lead to a safe or unsafe state within a given probability. In [Torsten Koller and Krause, 2018], for example, using a given affine control law in combination with the ellipsoidal bounding model the authors were able to arrive at a provably δ-safe algorithm (δ-safe meaning that the actions the system takes are safe with probability $(1 - \delta)$).

An ellipsoid can be defined as:

$$E(p, Q) := \{x \in \mathfrak{R}^n | (x - p)^T Q^{-1} (x - p) \leq 1\} \tag{9}$$

where $p \in \mathfrak{R}^n$ is the *center* of the ellipsoid, $Q \in \mathfrak{R}^{n \times n}$ is the symmetric positive definite *shape matrix*.

Ellipsoids have the property of being invariant under affine subspace transformations, i.e for $A \in \mathfrak{R}^{r \times n}$ ($r \leq n$) with full rank and $b \in \mathfrak{R}^n$:

$$A \cdot E(p, Q) + b = E(Ap + b, AQA^T) \tag{10}$$

The ellipsoid can then be used when considering a system of type:

$$x_{t+1} = f(x_t, u_t) = \underbrace{h(x_t, u_t)}_{\text{prior model}} + \underbrace{g(x_t, u_t)}_{\text{model error}} \tag{11}$$

it can be seen that using the system in the form of equation (11) allows for the system uncertainties to be included in the calculations, $h(x_t, u_t)$ is the *a priori* known estimate of the system and $g(x_t, u_t)$ is the error between this prior model and the actual model of the system.

As can be seen by equation (11), the system is required to be in discrete form to be used for ellipsoid calculations. Therefore, whenever ellipsoids are used, the state space matrices that are referred to and used are the discrete time versions of the continuous time systems, the time step used to discretize the matrices is the time step used in each simulation and varies according to the task (this information is contained in section V).

A. Using Ellipsoids for multi-step prediction

The approach presented in Torsten Koller and Krause [2018] uses Gaussian Processes (GPs) as a way to estimate the model error of the ellipsoids ($g(x_t, u_t)$) and thus adjust the ellipsoid and propagate them for several-step-ahead prediction. A gaussian process is fully defined by a mean function $m : \mathfrak{R}^d \to \mathfrak{R}$ and a covariance function $k : \mathfrak{R}^d \times \mathfrak{R}^d \to \mathfrak{R}$. This model error can be learned using from data obtained while running the algorithm and is then given as a Gaussian $\mathcal{G}(\mu_n(z), \sigma_n^2(z))$, where $z_t = (x_t, u_t)$. The mean and variance can then be calculated:

$$\mu_n(z) = k_n(z)^T [K_n + \lambda^2 I_n]^{-1} \tilde{y_n} \tag{12}$$

$$\sigma_n^2(z) = k(z, z) - k_n(z)^T [K_n + \lambda^2 I_n]^{-1} k_n(z) \tag{13}$$

$$[K_n]_{ij} = k(z_i, z_j); \qquad [k_n(z)]_j = k(z, z_j) \tag{14}$$

where $\tilde{y_n} = [y_1 - h(z_1), \cdots, y_n - h(z_n)]^T$ corresponds to the differences between the prior model h and the observed system response y at input locations $Z = [z_1, \cdots, z_n]^T$. The parameter λ^2 corresponds to the variance of the λ-sub-Gaussian sensor noise affecting the system (described as a normal distribution, $N(0, \lambda^2)$) that, in the case of the systems presented in this paper, is null due to the fact no sensor noise is considered.

The fact that GPs are used to estimate the model error makes this approach appealing to applications where the interval bounding model can't be easily determined. Initially n_0 samples are taken from within the safe region of the state space, i.e n_0 actions are taken from the initial position as a way to obtain an estimation from the model error with the GP.

Using ellipsoids presents some benefits when it comes to propagating predictions to multi-steps ahead. This is because, given an initial ellipsoid $R_0 \subset \mathfrak{R}^{n_x}$ and control input $u_t \in \mathcal{U}$ the confidence ellipsoids can be iteratively computed using:

$$R_{t+1} = \tilde{m}(R_t, u_t) \tag{15}$$

$\tilde{m}(R_t, u_t)$ corresponds to the over-approximation of the arrival set when action u_t is applied from departure set R_t.

The over-approximation (R_+) can be defined as:

$$R_+ = \bar{m}(R, u) = \tilde{f_\mu}(R, u) \oplus E(0, Q_{\bar{d}}(R, u)) \tag{16}$$

where $\oplus$ denotes the *Minkowski sum* and $\tilde{f}_\mu(R,u)$ is defined as:

$$\tilde{f}_\mu(R,u) = E(h(\bar{z}) + \mu(\bar{z}), AQA^T) \tag{17}$$

where $\bar{z} = (p,u)$ and $E(0, Q_{\tilde{d}}(R,u))$ is the approximation of the hyper-rectangle $\tilde{d}(R,u)$:

$$\tilde{d}(R,u) \doteq \beta_n \sigma_{n-1}(\bar{z}) + \frac{L\nabla_h l^2(R,u)}{2} + L_g l(R,u) \tag{18}$$

$$0 \pm \tilde{d}(R,u) \subset E(0, Q_{\tilde{d}}(R,u)) \tag{19}$$

$$a \pm b \subset E(a, \sqrt{n_x} \cdot diag([b_1, \ldots, b_{n_x}])) \tag{20}$$

where $l(R,u)$ can be defined as $l(R,u) = \max_{x \in R} ||x - p||_2$, i.e the furthest away point from the center of the ellipsoid that still belongs to it. Equation 20 describes how matrix $Q_{\tilde{d}}$ can be calculated by using an example with two random n-sized vectors a and b, n_x corresponds to the size of these vectors. $L\nabla_h$ is the Lipschitz constant of the gradient ∇h and β_n is a bound on the model error that can be estimated by:

$$\beta_n = B_g + 4\lambda\sqrt{K} \tag{21}$$

where B_g is described by $B_g \geq ||g||_k$. The second term of equation 21 is related to sensor noise and, thus, the value K is irrelevant for the applications shown in this research ($\lambda = 0$, since no sensor noise is considered), for other applications [Torsten Koller and Krause, 2018] presents a way to estimate K.

Equation 16 can then be over-approximated using the properties described in [Torsten Koller and Krause, 2018], namely using:

$$E(p_1, Q_1) \oplus E(p_2, Q_2) \subset E(p_1 + p_2, (1 + c^{-1})Q_1 + (1 + c)Q_2) \tag{22}$$

where $c > 0$ is a constant that can be taken as $c = \sqrt{\frac{Tr(Q_1)}{Tr(Q_2)}}$ so as to minimize the trace of the resulting shape matrix.

An example of the overall result of the multi-step-ahead prediction using ellipsoids can be seen in figure 4, here it is visible the over-approximating aspect of the ellipsoid bounding model.

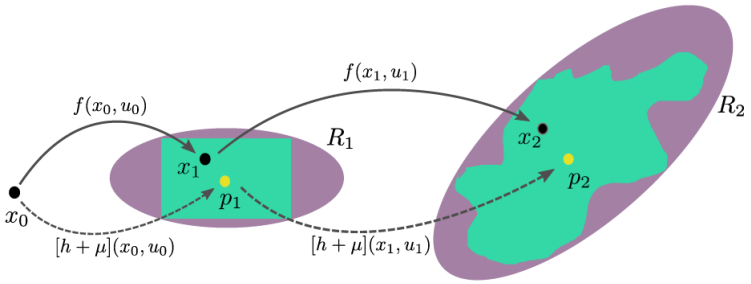


Figure 4 Ellipsoid multi-step-ahead prediction (taken from [Torsten Koller and Krause, 2018])

As can be seen from figure 4 the ellipsoid bounding model gives a much more realistic over-approximation of the arrival set thanks to its appealing statistical and geometric properties, it is much closer to the behavior one would expect of the system when compared with, for example, the interval bounding model which produces a hyper-rectangle. If one wants to calculate if a point is within this arrival set or not then one need only verify if it complies with equation 9, this is an appealing property since the arrival set can be very irregular in shape and complex to describe mathematically.

IV. Adaptations to the algorithm

In order to use the metrics described in section II.A in combination with the ellipsoidal bounding model some adaptations have to be made as now the interval analysis method is no longer used.

A. Proximity metric

The adaptation of the proximity is very straightforward, since the calculation of the ellipsoid at the next time step includes the calculation of the new center of the ellipsoid, which is itself the center of the arrival set collection. Therefore for the proximity metric the only thing that is required to calculate is the second term in equation (5), to determine this value one can take into account that the eigenvalues of the shape matrix (Q) of the ellipsoid corresponds to the length of each of its semi-axis. However, since this distance has to be weighted through the v_r vector, the largest eigenvalue of the Q matrix might not correspond to the largest distance from the center, and thus, what can be done is to apply a "stretching" transformation to the ellipsoid so that its value will consider this weighting. This stretching transformation is done through the S matrix:

$$\mathbf{S} = \begin{bmatrix} v_{r_1} & 0 & 0 & 0 \\ 0 & v_{r_2} & 0 & 0 \\ 0 & 0 & v_{r_3} & 0 \\ 0 & 0 & 0 & v_{r_4} \end{bmatrix} \tag{23}$$

The new ellipsoid is then considered with matrix:

$$Q_{new} = SQS^T \tag{24}$$

The maximum eigenvalue of the matrix can then be calculated and turned into the corresponding distance, in tiles, to be used as the second term in equation 5.

B. Operative metric

When it comes to the operative metric, since it is based on averaging the value of the tiles belonging to the arrival set, a way to find the "edges" of the ellipsoids has to be used. For this, the information that the eigenvalues of the shape matrix correspond to the length of each of the ellipsoid's semi-axes is used. The only adaptation that is needed is to, thus, calculate the corresponding eigenvalues of the shape matrix and use those as an over-approximation of the tiles the ellipsoid reaches. It is then a matter of using the metric as described and calculating the value of the arrival set.

V. Experimental Setup

In order to benchmark the different algorithms and study which combination of the methods described in the previous sections would be more optimal several experiments are conducted.

	Operative	Proximity	ProxOp
Quadrotor Navigation	E1	E2	E3
Elevator Control	E4	E5	E6

Table 1 Experiments to be conducted

In a first stage there will be six experiments to be conducted as shown in table 1. These experiments will test the three metrics in two different settings so as to compare overall performance.

In a second stage the ellipsoids will be included into the experiments as a replacement for the bounding model which will originate three more experiments, as the ellipsoids only work on continuous time models and the quadrotor task includes a discrete state. This second stage will test whether the ellipsoids are a suitable replacement for the bounding model used previously. Ultimately the improvements that are sought after are in terms of overall safety (survivability of the algorithm during the simulation) and efficiency (measured by different run time values).

The models used in the different experiments are thoroughly described in [Mannucci, 2017] and are reproduced here.

- **For the quadrotor navigation task:**

$$\mathbf{x} = \{x_1, x_2, V, \psi, \theta\} \tag{25}$$

$$\dot{x}_1 = Vcos(\psi); \quad \dot{x}_2 = Vsin(\psi); \tag{26}$$

$$\dot{V} = \theta \dot{V}_c; \quad \mathcal{A} = \{\texttt{forw}, \texttt{back}, \texttt{right}, \texttt{left}, \texttt{neut}\} \tag{27}$$

$$\dot{\phi} = \begin{cases} +\dot{\phi}_c & if\ \texttt{right} \\ -\dot{\phi}_c & if\ \texttt{left} \\ 0 & if\ \texttt{neut} \end{cases} \qquad \Delta\theta = \begin{cases} +1 & if\ \texttt{forw} \wedge \theta \neq +1 \\ -1 & if\ \texttt{back} \wedge \theta \neq -1 \\ 0 & if\ \texttt{neut} \end{cases} \tag{28}$$

Parameter	Value	units
$x_{1\|0}$	0	[m]
$x_{2\|0}$	0	[m]
$V_{\|0}$	$\in [0.4, 0.6]$	[m/s]
$\phi_{\|0}$	$\in [-\pi, \pi]$	[rad]
θ_0	0	[rad]
$\hat{\dot{V}}_c$	$[0.24, 0.6]$	[m/s^2]
$\hat{\dot{\phi}}_c$	$[\pi/4, \pi/3]$	[sec^{-1}]
Δt (time-step)	0.5	[sec]
x_{detect} (detection range)	2.5	[m]
k (action length)	$\in \{3, 4, 5\}$	[-]

Table 2 Parameters for the quadrotor navigation task.

where $\mathbf{x}$ is the state vector.
In addition to the equations of motion described above and the parameters in table 2, some additional information about the quadrotor task is required. Namely, the drone is considered to be operating within a 10×10 meter room and an episode is considered successful if the system reaches the 300th time step without colliding with a wall. The values for the two model parameters $\dot{\phi}_c$ and $\dot{V}_c$ are randomly assigned at the start of each episode, $\dot{\phi}_c \in \hat{\dot{\phi}}_c$ and $\dot{V}_c \in \hat{\dot{V}}_c$. The bounds for the different states are defined as: $x_1, x_2 \in [-5, 5]$m, $\phi \in]-\pi, \pi]$rad, θ is discrete with possible values of $\theta = \{-1, 0, 1\}$ and the velocity is artificially restricted to be bounded as $V \in [-1.2, 1.2]$m/s. Tiles are then generate by dividing each state into 20 equally-sized tiles.
For the operative metric the parameters used are: $q_{exp} = 1$, $q_{safe} = 0$, $q_{unc} = -100$ and $q_{fat} = -10^6$. For the proximity metric the parameters used are: $\mathbf{v_r} = (5, 5, 2, 1, 1)$ and $\rho = 0.3$.

- **For the elevator control task:**

$$\begin{bmatrix} \dot{h} \\ \dot{\theta} \\ \dot{\alpha} \\ \dot{q} \end{bmatrix} = A \begin{bmatrix} h \\ \theta \\ \alpha \\ q \end{bmatrix} + B_n \delta_e; \quad A = \begin{bmatrix} 0 & 300 & -300 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & -0.64 & 0.938 \\ 0 & 0 & -1.568 & -0.879 \end{bmatrix}; \quad B_n = \begin{bmatrix} 0 \\ 0 \\ B^{\alpha}_{\delta_e} \\ B^{q}_{\delta_e} \end{bmatrix} \tag{29}$$

where $B^{\alpha}_{\delta_e} = -1.4 \cdot 10^{-3}$ and $B^{q}_{\delta_e} = -0.1137$. As with the quadrotor navigation task the actual dynamics differs from the nominal mechanics, the bounding model for the system is, thus, $\hat{B} = [1.05 \cdot B_n, 0.95 \cdot B_n]$ and the actual dynamics are randomly selected at the start of the episode ($B \in \hat{B}$). The action set for the elevator control task is defined as $\mathcal{A} = \{-4, -2, 2, 4\}$.
In addition to the equation of motion described above and the parameters defined in table 3, some additional information is required to set-up the simulation. In this application an episode is considered to be successful if it manages to stay within the prescribed bounds of altitude and angle of attack for 600 iterations. The bounds for the different states are defined as: $h \in [-80, 80]$ft, $\alpha \in [-15, 12]$, θ and q are artificially bounded between

Parameter	Value	units
h_0	0	[m]
θ_0	0	[rad]
α_0	0	[rad]
q_0	0	[rad/s]
Δt (time-step)	0.1	[sec]
h_{detect} (detection range)	30	[ft]
α_{detect} (detection range)	$\frac{\pi}{30}$	[rad]
k (action length)	$\in \{3, 4, 5\}$	[-]

Table 3 Parameters for the elevator control task.

$[-\pi/4, \pi/4]$ and $[-\pi/2, \pi/2]s^{-1}$, respectively.

As mentioned in section III, the time-step value used for discretizing the state space of the elevator control task for use with the ellipsoid bounding model is the one shown in table 3.

VI. Results and Discussion

Experiments E1,E2,E4 and E5 (as described in table 1) were replicated from those conducted in [Mannucci, 2017] in order to establish a benchmark to compare the performance of the new algorithms to.

The main measure of performance of these algorithms is related to the duration of the runs (number of iterations) since it is directly related to the survivability of the agent, as such the average duration of the episode, in iterations, directly relates to how survivable the system is with each algorithm. Further measures of performance that were used include the success rate and the run time. In order to account for the fact that algorithms that have a lower success rate will also show a reduced run time, due to the fact that failing an episode means that less iterations have been concluded than if the system finished the episode, the main run time measurement used is run time per iteration as this accounts for the difference generated by the different survivability.

All results shown in this section are averaged over 100 episodes of the algorithms.

A. Results using the interval bounding model

In this subsection the results for the 3 metrics using the previous interval bounding model are shown. Tables 4 and 5 shown the average duration of an episode (as a measure of survivability) and the success rate (as a further measure of performance), for both tasks.

Looking first at table 4, it is clear that the Operative metric is more adequate for the quadrotor navigation task when compared with the proximity metric, this is evidenced both by the average duration and the success rate of the episodes. It is also clear that the ProxOp metric, although performing worse than the Operative metric, performs significantly better than the proximity metric in the task.

Secondly, when the data shown in table 5 is analyzed it becomes clear that for the elevator control task now the proximity metric is the one that is clearly superior in regards to the operative metric. Another important result is that the ProxOp metric has near identical performance when compared to the proximity metric.

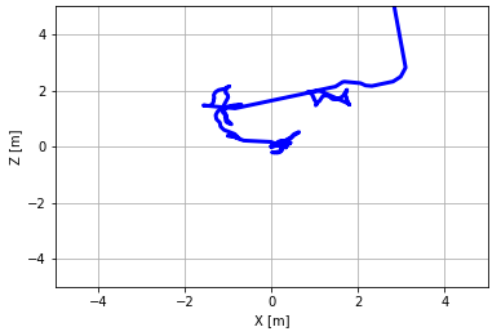
From these sets of data some results can be extracted:

- The Operative and Proximity metrics are better than one another at different tasks.
- The ProxOp metric is better than the worse one of the two other metrics in both tasks.
- In one of the tasks theProxOp metric has performance comparable to the best of the metrics.

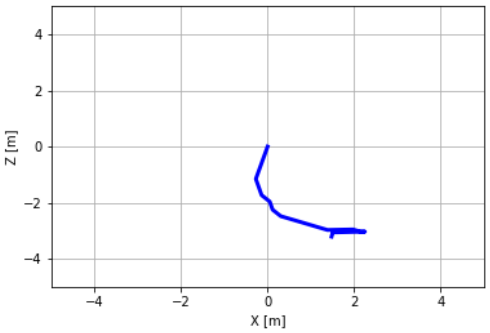
It can be argued, then, that in case no previous information is know about which metric will work best, that the ProxOp metric is the best option to go with. This is because it is going to provide a better performance than the worst case scenario and it might even perform close to the best case scenario. These results, thus, support the assumption that the combination of the two metrics is valid and present suitable performance.

Metric	Average Duration of episode (Iterations)	Success rate [%]
Operative Metric	296.11	99%
Proximity Metric	69.03	7%
ProxOp Metric	143.02	28%

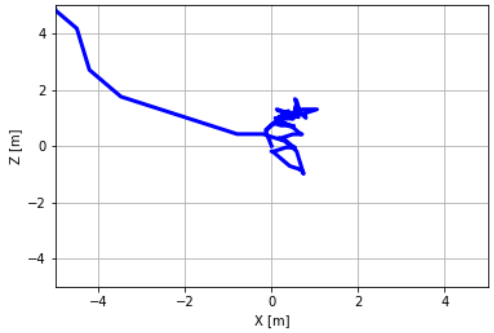
Table 4 Duration in quadrotor task (iterations), averaged over 100 runs, interval bounding.



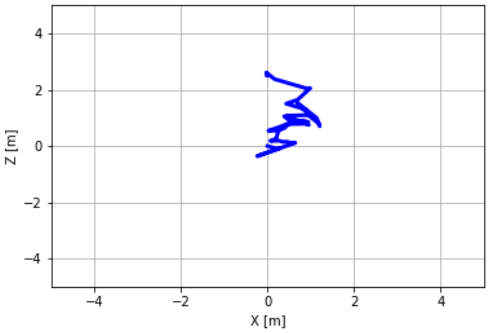
(a) Operator metric, example of an successful run.



(b) Proximity metric, example of a successful run



(c) ProxOp metric, example of an unsuccessful run.



(d) ProxOp metric, example of a successful run.

Figure 5 Examples of a few runs of the quadrotor navigation task with different metrics using the interval analysis bounding model.

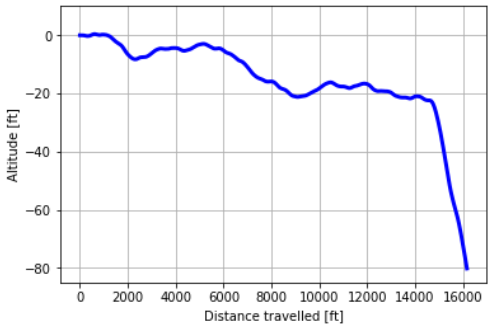
Metric	Average Duration of episode (Iterations)	Success rate [%]
Operative Metric	443.53	54%
Proximity Metric	570.5	90%
ProxOp Metric	560.63	88%

Table 5 Duration in elevator control task (iterations), averaged over 100 runs, interval bounding.

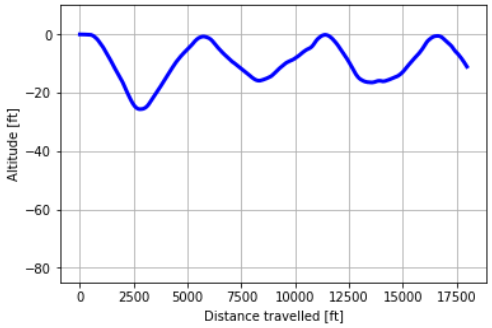
B. Results using the ellipsoid bounding model

In table 6 results are shown for the elevator control task of the different metrics using the ellipsoid bounding model. One of the comparisons that can be made is between these results and the ones shown in table 5. Some results stand out immediately:

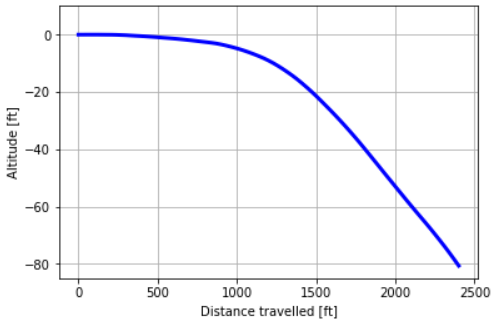
- The ellipsoid model degrades the performance of the operative metric considerably when compared to the previous test with the interval analysis bounding model.



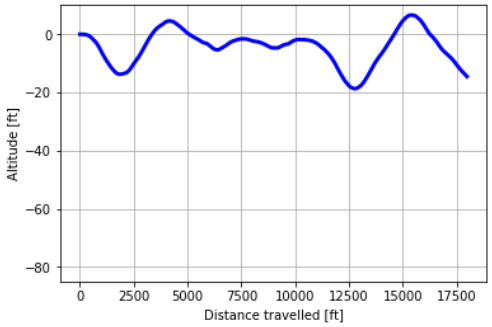
(a) Operator metric, example of an unsuccessful run.



(b) Proximity metric, example of a successful run



(c) ProxOp metric, example of an unsuccessful run.



(d) ProxOp metric, example of a successful run.

Figure 6 Examples of a few runs of the elevator control task with different metrics using the interval analysis bounding model.

- The ellipsoid model also degrades the performance of the two other metrics, albeit in a much smaller capacity. The two metrics still achieve very high performance, although their success rates are reduced slightly. The average duration of the episodes actually goes up.

From these results it is clear that the ellipsoid bounding model is suitable for use, as long as it is used with the proximity metric or the ProxOp metric as it degrades the operative metric performance significantly. Figure 7 shows some examples of runs conducted by these various metrics, the actions taken in each run vary but it was seen during the simulations that in the case of the operator metric the system was often unable to recover from a climb or descent.

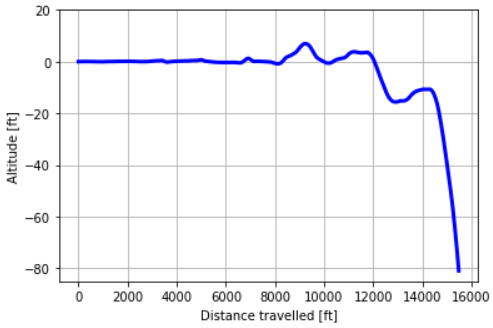
Metric	Average Duration of episode (Iterations)	Success rate [%]
Operative Metric	340.41	11%
Proximity Metric	574.1	85%
ProxOp Metric	562.75	81%

Table 6 Duration in elevator control task (iterations), averaged over 100 runs, ellipsoid bounding.

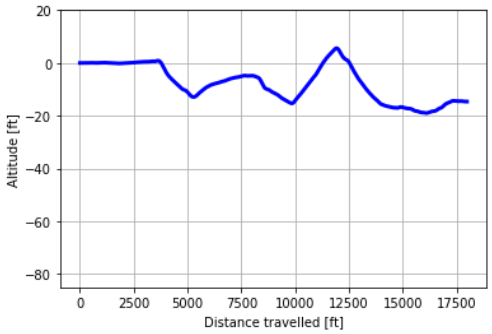
C. Run time results

Another important factor when considering the performance of the algorithms is, as discussed previously in this section, the computational efficiency of the algorithm. This efficiency can be determined by considering the run time of each algorithm, when considering the same hardware was used for simulations. The hardware used for the simulations is made up of a Intel $i5-8250U$ CPU and 8Gb of RAM memory.

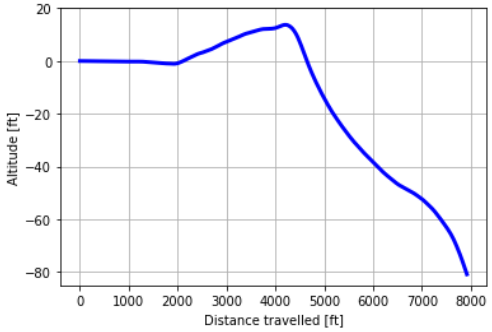
Table 7 shows the run time results for the elevator control task using the two different bounding models and the three



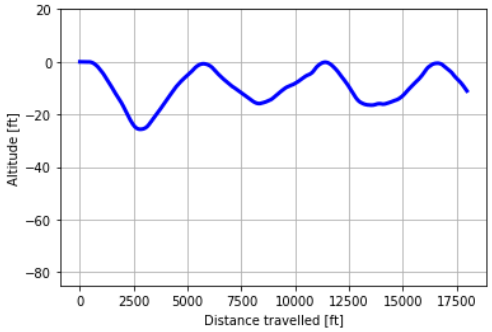
(a) Operator metric, example of an unsuccessful run.



(b) Proximity metric, example of a successful run



(c) ProxOp metric, example of an unsuccessful run.



(d) ProxOp metric, example of a successful run.

Figure 7 Examples of a few runs of the elevator control task with different metrics using the ellipsoid bounding model.

different metrics.

Metric	Bounding model	Total run time [sec]	Run time per iteration [sec]
Operator	Interval Analysis	14824.39	0.334
Proximity	Interval Analysis	53.77	0.000656
ProxOp	Interval Analysis	41238.82	0.736
Operator	Ellipsoid	16796.23	0.493
Proximity	Ellipsoid	559.44	0.0095
ProxOp	Ellipsoid	27360.35	0.486

Table 7 Run time results for the Elevator Control Task, averaged over 100 runs.

The most clear conclusion is that the proximity metric is the most efficient of the three metrics, being several orders of magnitude faster in terms of time per iteration than both the operator and ProxOp metrics. The ProxOp and operator metrics have similar run times since the biggest computational drain occurs when updating the safe state space, for that calculation a nested *for* cycle is used to update all the values of the visible states from q_{exp} to either q_{safe}, q_{exp} or q_{fat}, this computation takes up a lot of run time since every tile that is visible has to be checked to see if their value is to be updated or not (a tile that had a previously assigned value of q_{safe}, for instance, cannot have its value changed to q_{fat}). Since this part of the process has to be done both in the Operator and ProxOp metrics both have long run times, when compared to the proximity metric.

Another aspect to notice is that using the ellipsoids did not reduce the run time of the algorithms it, in fact, increased the run time. Which means that, adding to the conclusions drawn in section VI.B, the ellipsoid causes the system to have worsen its computational efficiency.

D. Discussion of the results

The results gathered in the previous 3 subsections point towards certain conclusions being drawn. They shall now be restated:

- The ProxOp metric is a better option for use when it is not known which metric will operate better for the task, this is justified by the fact its performance is better than the worst case scenario and can even be close to the best case scenario.
- Using the ProxOp metric does not significantly increase computational time, when compared to the operator metric. Both the Operator metric and the ProxOp metric show a run time several orders of magnitude higher than the proximity metric. Most of this increased run time is expended in updating the values of the tiles.
- The ellipsoid bounding model degrades the performance of both the ProxOp and proximity metrics slightly but the operator metric shows a significant degradation in performance when used with it. It is, therefore, better not to use the operator metric with the ellipsoid bounding model.
- The ellipsoid bounding model increases the run time of the algorithm slightly in the case of all three metrics. This increase is not significant enough to be a deciding factor in its implementation unless the computation capacity is severely limited.

One conclusion that is not drawn from the data directly but that has to do with the way the ellipsoidal bounding model has been implement is that this bounding model is preferable when there is little information about the system. Since the performance degradation it causes is low when it comes to the ProxOp metric and since it does not require the knowledge of the bounds of the system model but instead estimates the model error through Gaussian processes, the ellipsoid bounding model can be used in cases where there might only be an estimate of the system model and no information as to what bounds the system model is within. Additionally, the fact that the computation time while using this bounding model does not increase significantly reinforces this conclusion

VII. Conclusions

This work has explored how to expand the existing methods used for SafeRL control so as to contribute towards using SafeRL control frameworks in several real world tasks, such as for the control of UAVs, by combining their ability to function model-free and to learn *"on the fly"* with methods to assure a safe exploration of the state space.

Previous literature had seen the development of two different safety metrics that could be used to achieve safe exploration, the work developed here has expanded upon those two metrics by combining them into the ProxOp metric which has been proven to also have adequate performance. Furthermore, the ProxOp metric is shown to be a viable alternative for use when no *a priori* information regarding which of the two previous metrics would have better performance is known.

Additionally, an ellipsoidal bounding model was combined with the metrics discussed as an alternative to the, previously used, interval analysis bounding model. This model does display significant degradation of performance when used with the operative metric but has been proven to lead to similar performance when used with both the Proximity and ProxOp metric, when compared to the interval analysis bounding model. The ellipsoidal bounding model is specially useful when considering that it does not require *a priori* information on the "bounds" of the model since the model error is calculated during the initial stages of the simulation and, thus, it requires only an approximate initial model and will determine the actual error, whereas the bounding model required bounding information on what the model error could be.

When it comes to expanding the research described here the focus should be on two main topics. Firstly increasing the safety of the algorithms, for example, by doing further step ahead prediction than the one carried out here. Although the results were satisfactory, the safety levels required for commercial usage are higher than the ones currently presented by the algorithms. Secondly, the operator and ProxOp metrics are computationally demanding to the point that they cannot be used for real time calculations which means that, in their current state, they cannot be used online (especially on a system like a UAV that has limited computational resources), one appealing prospect would be to find a more efficient way to update the values of the tiles since this is the part of these algorithms that uses up the most computational resources.

References

Sutton, R. S., and Barto, A. G., *Reinforcement Learning:An Introduction*, 2nd ed., The MIT Press, Cambridge, Massachusetts, 2017.

García, J., and Fernández, F., "A comprehensive survey on safe reinforcement learning," *Journal of Machine Learning Research*, Vol. 16, 2015, pp. 1437–1480.

Horie, N., Matsui, T., Moriyama, K., Mutoh, A., and Inuzuka, N., "Multi-objective safe reinforcement learning: The relationship between multi-objective reinforcement learning and safe reinforcement learning," *Artificial Life and Robotics*, 2019. Article in Press.

Martínez, D., Alenyà, G., and Torras, C., "Safe robot execution in model-based reinforcement learning," *IEEE International Conference on Intelligent Robots and Systems*, Vol. 2015-December, 2015, pp. 6422–6427.

Li, Z., Kalabić, U., and Chu, T., "Safe Reinforcement Learning: Learning with Supervision Using a Constraint-Admissible Set," *Proceedings of the American Control Conference*, Vol. 2018-June, 2018, pp. 6390–6395.

Isele, D., Nakhaei, A., and Fujimura, K., "Safe Reinforcement Learning on Autonomous Vehicles," *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS 2018, Madrid, Spain, October 1-5, 2018*, 2018, pp. 1–6. https://doi.org/10.1109/IROS.2018.8593420, URL https://doi.org/10.1109/IROS.2018.8593420.

Berkenkamp, F., Moriconi, R., Schoellig, A. P., and Krause, A., "Safe learning of regions of attraction for uncertain, nonlinear systems with Gaussian processes," *2016 IEEE 55th Conference on Decision and Control, CDC 2016*, 2016, pp. 4661–4666.

Luo, B., Liu, D., and Wu, H. ., "Adaptive Constrained Optimal Control Design for Data-Based Nonlinear Discrete-Time Systems with Critic-Only Structure," *IEEE Transactions on Neural Networks and Learning Systems*, Vol. 29, No. 6, 2018, pp. 2099–2111.

Mannucci, T., "Safe Online Robust Exploration for Reinforcement Learning Control of Unmanned Aerial Vehicles," Ph.D. thesis, Delft University of Technology, 2017.

Torsten Koller, M. T., Felix Berkenkamp, and Krause, A., "Learning-based Model Predictive Control for Safe Exploration," *IEEE Conference on Decision and Control*, 2018. Article in Press.

Filippova, T. F., "Estimates of reachable sets of a nonlinear dynamical system with impulsive vector control and uncertainty," *Proceedings of 2018 14th International Conference Stability and Oscillations of Nonlinear Control Systems (Pyatnitskiys Conference), STAB 2018*, 2018, pp. 1–4.

Asselborn, L., Groß, D., and Stursberg, O., "Control of uncertain nonlinear systems using ellipsoidal reachability calculus," *IFAC Proceedings Volumes (IFAC-PapersOnline)*, Vol. 46, 2013, pp. 50–55.

3

Reinforcement Learning

This chapter will first begin with a quick introduction to some Reinforcement learning concepts and methods and will then finish with a review of the current state of the art when it comes to RL (with a special focus on research that addresses the problem of SafeRL).

3.1. Reinforcement Learning: some basic concepts

In 1.1 and 1.2 a very brief introduction to RL was given, in this section a more in depth look will be taken at RL and at some of its methods and concepts.

The first thing that has to be defined when talking about RL is the **agent-environment interaction**, as shown in figure 3.1. In RL the agent is the object or machine that is to be controlled and the environment is what's around such machine, for example if a robot was placed in a 2-D gridworld the agent would be the robot (that could move in the 4 cardinal directions) and the environment would be the gridworld. The agent takes an **action** (A_t) and obtains a **reward** (R_t) and information on the **state transition** (S_{t+1}).

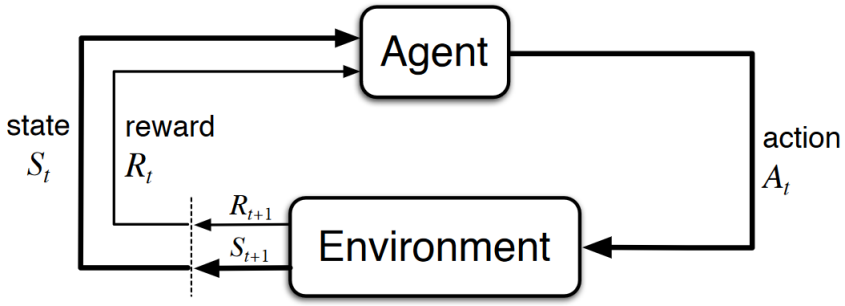


Figure 3.1: The agent-environment interaction in RL, taken from [Sutton and Barto, 2017]

The goal in RL is always to maximize a reward (or return if discounted rewards are used), RL is then an optimization problem. A return is merely an expected reward of a given state that takes into account not only the reward received when transitioning to that state but also the expected reward of the subsequent states.

In order to maximize the reward, the agent must then find an optimal **policy** (π, that basically tells the agent which action to chose in each state) that will maximize the reward. Finding a policy amounts to saying that a mapping has been found that describes the probability of selecting each action in each state.

The way the agent evaluates which states are best is through its **value function**, almost all RL methods involve some sort of value function estimation. This is because the agent rarely has information so as to the real value of the cost function at the start of the problem, it must then estimate what its value might be. The "value" of a given state is merely a measure of what the future rewards, from that state onward, can be expected, in other words, what the expected return of a given state is.

There are two different types of value functions that are used by different RL methods and algorithms:

[0] **This chapter has already been graded for AE4020.**

- The state value function, $\mathbf{v}_\pi(\mathbf{s})$, that is a measure of the expected return if the agent is currently in state **s** and is going to follow policy π from then on, and is formally defined as:

$$v_\pi(s) = \mathbb{E}_\pi\Big[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \mid S_t = s\Big] \tag{3.1}$$

  where $\mathbb{E}$ is the mathematical symbol for expectation, γ is the discounting factor, R_t is the reward at time t and S_t is the state at time t.

- The state-action value function, $\mathbf{q}_\pi(\mathbf{s},\mathbf{a})$, that is a measure of expected return of taking action **a** in state **s** and then following policy π, and is formally defined as:

$$q_\pi(s,a) = \mathbb{E}_\pi\Big[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \mid S_t = s, A_t = a\Big] \tag{3.2}$$

The optimal policy (π_*) is then defined as the policy that maximizes the value function. This optimal value function can then be defined for the state value function:

$$v_* \doteq \max_\pi v_\pi(s) \tag{3.3}$$

And for the case the state-action value function:

$$q_* \doteq \max_\pi q_\pi(s,a) \tag{3.4}$$

for all $\mathbf{s} \in \mathbb{S}$ and $\mathbf{a} \in \mathbb{A}$.

Because they are optimal values, both $\mathbf{v}_*$ and $\mathbf{q}_*$ can be rewritten in a special form, in the form of **Bellman optimality equations**:

$$v_*(s) = \max_a \sum_{s',r} p(s',r \mid s,a)\big[r + \gamma v_*(s')\big] \tag{3.5}$$

$$q_*(s,a) = \max_a \sum_{s',r} p(s',r \mid s,a)\big[r + \gamma \max_{a'} q_*(s',a')\big] \tag{3.6}$$

Where $p(s',r \mid s,a)$: is the transitioning function, that gives the probability of transition to state s' and receiving reward r when the agent is at state s and takes action a. Equations 3.1 to 3.6 are fully explained in [Sutton and Barto, 2017].

These equations can also be graphically represented by backup diagrams (figure 3.2), backup diagrams give us the information as to the span of future actions and states that the Bellman equation considers.

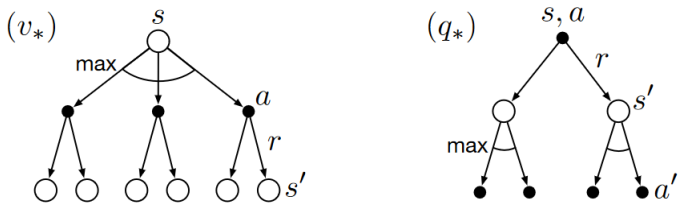


Figure 3.2: Backup diagram for both v_* and q_*, taken from [Sutton and Barto, 2017]

Finding the optimal policy through explicitly solving the Bellman equation amounts to exhaustively looking at all the possible state transitions until the episode is over from each state. This search can be extremely time and computationally expensive, it also assumes that the system follows the Markov property and that a complete and reliable model of the system is available. It's quite clear why only on rare occasions the Bellman equation is solved explicitly:

- There might be some limitations in computational resources or the amount of possibilities might be too large for a computer to explore in useful time
- The only system model might be inaccurate, or there might not be any model at all available
- The system might not follow the Markov property

For real-life problems it is usually too expensive computationally to exhaustively search the state space in order to get an optimal policy from the Bellman equation. Therefore, in almost all applications what the different RL methods try to achieve is finding a policy that is approximately optimal.

The way these approximations are computed also has to be thought off, if simply a table is used (for example) to store all the approximate values of each state, the problem might have an enormous amount of states which might make storing in memory each approximate value of each state impossible. This can be mitigated by using function approximation methods such as splines and neural networks, that way only a certain number of parameters have to be stored in memory and continuously updated every-time the agent gets a new value of the value function.

3.2. Some RL methods

In this section two different RL methods will be discussed as examples of usage of the concepts described in 3.1.The two methods that will be discussed are **SARSA** and **Q-Learning**, which are two different types of Temporal-Difference (TD) methods.

TD methods use experience as a way to solve the prediction problem, they use the experience they gained following policy π to update their estimation of their estimation of the value function (V) at each time step, at time $t+1$ they use the value of R_{t+1} and the estimate of the value function at state S_{t+1} ($V(S_{t+1})$ to update the current estimate of the value function. The update law they use depends on the method itself and on how many steps ahead the agent looks.

3.2.1. SARSA

SARSA is an on-policy TD control method that uses an state-action value function. The update law for the current estimate of the value function is given by ([Harm van et al., 2009]):

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha\big[R_{t+1} + \gamma Q(S_{t+1}, A_{t+1}) - Q(S_t, A_t)\big] \tag{3.7}$$

thus, the name SARSA: State, Action, Reward, State, Action (the information needed for the update).

If an agent is to be trained using SARSA, algorithm 1 (taken from [Sutton and Barto, 2017]) can be used.

Algorithm 1 SARSA, online TD algorithm

Initialize: Q(s,a) arbitrarily except Q(terminal state,-)=0
repeat *for each episode*:
    Initialize S
    Choose A from S using policy derived from Q (for example, $\epsilon - greedy$
    repeat*for each step of the episode*:
        Take action A, observe reward R and transition S'
        Choose A' from S' using policy derived from Q
        $Q(S, A) \leftarrow Q(S, A) + \alpha\big[R + \gamma Q(S', A') - Q(S, A)\big]$
        $S \leftarrow S'; A \leftarrow S'$
    until S is terminal
until maximum number of episodes is reached

3.2.2. Q-Learning

Q-Learning is another TD method similar to SARSA (with some differences, including being an offline method).

The following update law is now used ([Watkins and Dayan, 1992]):

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha\big[R_{t+1} + \gamma \max_a Q(S_{t+1}, a) - Q(S_t, A_t)\big] \tag{3.8}$$

As with SARSA, an algorithm can be defined to train an agent for a given task using Q-learning, this is described in algorithm 2 (taken from [Sutton and Barto, 2017]).

Algorithm 2 Q-learning, offline TD algorithm

Initialize: Q(s,a) arbitrarily except Q(terminal state,-)=0
repeat *for each episode*:
    Initialize S
    repeat*for each step of the episode*:
        Choose A from S using policy derived from Q (for example, $\epsilon - greedy$
        Take action A, observe reward R and transition S'
        $Q(S,A) \leftarrow Q(S,A) + \alpha\left[R + \gamma \max_a Q(S',a) - Q(S,A)\right]$
        $S \leftarrow S'$
    until S is terminal
until maximum number of episodes is reached

3.2.3. SARSA vs Q-Learning

Although SARSA and Q-learning are similar methods in that they are both TD methods, their performance is very different. The example used to display their difference in performance is usually the cliff gridworld navigation problem. The problem is defined in [Sutton and Barto, 2017] by using figure 3.3, the reward is -1 at each step except if the agent falls into the cliff, then the reward is -100 and the agent's position gets reset to S and the episode continues. Each episode ends once the agent reaches G.

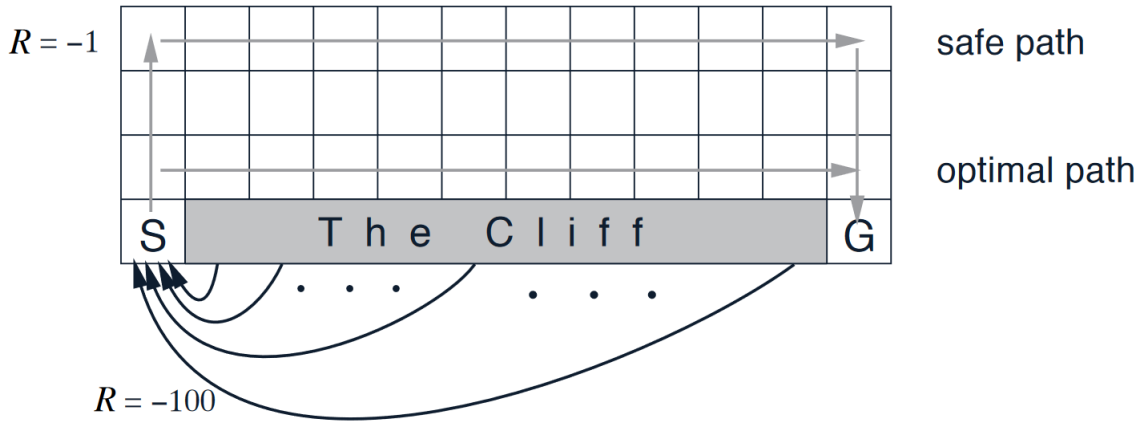


Figure 3.3: Cliff walking problem

Figure 3.3 (ϵ-greedy policy) also displays the two paths the algorithms converge to: SARSA tends to converge to the safer path while Q-learning tends to converge to the optimal (faster) path, this causes the agent to occasionally fall into the cliff due to the policy being ϵ-greedy which results in overall lower values in rewards. Of course if ϵ was asymptotically reduced (for example using a law of $\epsilon = \frac{1}{t}$) then both algorithms would converge to the same optimum.

3.2.4. *n*-step ahead prediction

Both methods that were just mentioned both involve 1 step ahead prediction, however there might be some interested to use methods that look further ahead (the limit would be to look at all the steps until the episode ends, like Monte Carlo methods do). One application would be to the Safe RL problem that is to be solved, if the part of the state-space where the aircraft will be in n-steps can be predicted it can also be predicted whether a certain action will eventually lead to an unsafe state or not.

There are natural extensions of both SARSA and Q-learning to n-step ahead prediction ([Sutton and Barto, 2017]).

The extension of the Q-learning (offline) to n-steps involves also using importance sampling, this is because often when offline n-step ahead prediction learning algorithms are used the objective is to learn a target policy from a given taken policy. Importance sampling is a way of scaling the results to reflect that some actions might be taken less or more often (or not at all) by the target policy with respect to the policy that was followed during the experiments.

SARSA does not require importance sampling because it is an online algorithm.

3.3. State of the Art in Reinforcement Learning

As is true with most fields relating to Machine Learning and Artificial Intelligence, there has been a great deal of papers and books produced about Reinforcement Learning in the past decade. One of the best pieces of literature to present a introduction to the world of RL and the basics of certain tried-and-tested algorithms (such as Dynamic Programming, Monte Carlo Methods, SARSA and Q-Learning) is [Sutton and Barto, 2017].

3.3.1. Reinforcement Learning in the Flight Control Domain

When discussing RL in the context of Flight Control it is important to mention that this domain is not new to the application of such algorithms. There has been several research efforts dedicated to different applications of RL to this domain.

In this subsection some of these applications will be discussed as a way of illustrating how RL is relevant towards this specific field of engineering.

With the rising interest in exploring (and even colonizing) Mars, it is important to consider whether the current guidance algorithms used for the landings are good enough or should be improved. The work done in[Gaudet and Furfaro, 2014] addresses this issue, the authors developed a RL-based adaptive guidance control trained to learn an optimal landing policy. Using apprenticeship learning (imitation learning) the authors use expert samples to initialize the Neural Network weights for the RL agent. The reward used was quadratic in nature for the deviation from the desired landing site and had an additional linear term to account for loss of weight due to fuel consumption. The agent was able to train to achieve good performance in guiding the lander near-optimally and it was even shown to be able to execute re-targeting (changing desired landing spot) while en-route. This research is relevant given the fact that the a-priori chosen spot for a landing might present unforeseen obstacles and, during landing, the lander might be required to make a change in trajectory to account for a change in desirability of landing locations.

In both [Zhang et al., 2017] and [Lee and Bang, 2010] actor-critic structures (the actor network controls the system and the critic network approximates the true value function) for RL are used.

In [Zhang et al., 2017] a steady-state controller is designed for aero-engine control that is shown to have strong disturbance rejection, adaptability to changes in the external environment and robustness in dealing with perturbations in the parameters of the aero-engine.

In [Lee and Bang, 2010] the same kind of structure is applied to a classic feedback controller. The RL agent compensates for the poor performance of an *a priori* designed sub-optimal PID controller and the system is shown to perform better than the PID controller without the RL agent. The system proposed is tested and achieves the required control performance after training for a task involving a nonlinear, complex representation of an unmanned helicopter.

In [Van Kampen et al., 2006] the actor-critic structure for a RL agent is again used, in it two different types of Actor Critic Designs (ACDs) are explore: an Heuristic Dynamic Programming (HDP) with access to an approximate model of the plant and a Action Dependent HDP (ADHDP) where the plant is *implicitly* stored by the critic. These two approaches are compared in an implementation for the F-16 aircraft and it is verified that the HDP with the approximate plant model supersedes the performance of the ADHDP in both offline and online training as well as having a wider range of flight conditions it can operate at (the ADHDP is found to have better performance with noise present).

In [Ferrari and Stengel, 2002], a dual heuristic adaptive critic approach is developed, this approach is based on an actor-critic architecture where both structures are represented by neural networks. Neural Networks are used because they easily handle high dimensional input. The resulting algorithm differs from traditional actor-critic architectures in that the critic is used to approximate the derivatives of the cost function instead of the cost function itself. A 2-stage process is involved where, in the first stage, the system is initialized using algebraic results of known operating points (based on an LQR controller), in the second stage the system performs online learning and the neural networks are used to account for differences between the assumed and the actual dynamics of the system using Adaptive Dynamic Programming (ADP). This modified algorithm is shown to have a better performance than a comparable `Matlab` algorithm, in terms of the amount of epochs needed to converge to an acceptably low error level, as shown in figure 3.4.

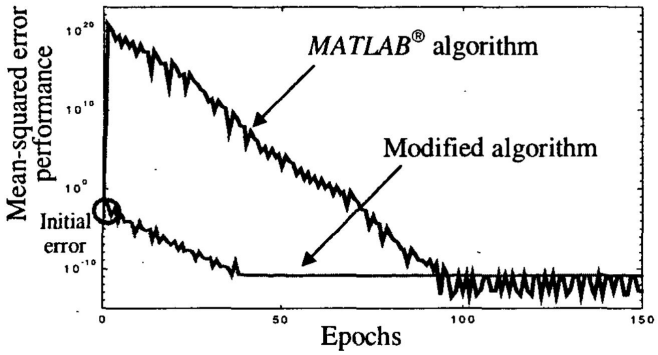


Figure 3.4: Comparison between the `Matlab` RPROP method and its modified version, taken from [Ferrari and Stengel, 2002]

[Fang et al., 2012] displays an application of an Actor-Critic framework, this time a episodic Natural Actor-Critic (eNAC) framework is developed. The main focus of this work is to develop a system that uses this eNAC framework to learn a Linear Parameter-Varying (LPV) controller for the longitudinal dynamics of an helicopter model. The reason this LPV controller is chosen is because the its gain-scheduled PID structure is useful in accounting for the nonlinear aspects of the model used. The LPV controller is shown to have better performance than the conventional PID controller in reference tracking for a situation of cruise flight of the nonlinear model. The LPV is based on a control law that is seemingly linear but is governed by a varying scheduling parameter that can either be measured or determined *a priori*. The algorithm starts from an initial state and policy and uses step-excitations, that are chosen so as to cover the full dynamical range of the system, as the reference signal in order to learn, the performance improves as the algorithm learns as shown by figure 3.5. Additionally a notch filter is used as, for this model, to handle the rotor-flapping and fuselage-pitching couplings present. The reward signal is generated using the reference model. Overall the system displays good tracking performance but due to the cross dependencies of the input and ouput it is unable to handle the MIMO system.

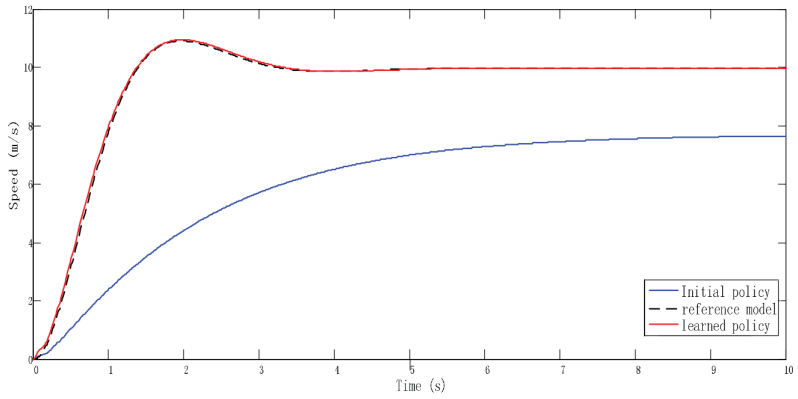


Figure 3.5: Performance comparison of learned and initial policies, taken from [Fang et al., 2012]

3.3.2. Safe Reinforcement Learning (SafeRL)

As was discussed in section 1.1, there have been quite some efforts undertaken in the field of RL when it comes to making exploration safe. This is motivated by the fact that for some tasks safety is absolutely **critical**. Almost all of the current and past literature has tried to achieve safety in exploration through either including it as a goal in the objective function or through constraining the safe state space. Although these two frameworks are usually used there are many different ways to implement them. In [García and Fernández, 2015] an in depth evaluation is done of both methods.

This section will present a revision on the current state of the art in Reinforcement Learning, and specifically in **Safe** Reinforcement Learning. It will be divided into three subsections: firstly the articles that present some way of state space constraints will be presented, then the articles that use safety as a part of the objective function and finally a subsection will be dedicated to miscellaneous articles that present interesting

concepts that might be used in conjunction with one of the SafeRL methods as a way to improve performance or that represent some field of RL.

SafeRL through state space constraints

In [Martínez et al., 2015] if an action is deemed to make the system transition to a risky state the agent looks for an alternative or, if it can't find one, asks for teacher advice. The system eventually learns the dangerous 'predicates' that might lead to risky states and avoids taking actions with a high enough probability of reaching these 'predicates'. This way, the state space is successively constrained to the subset that the agent knows has low probability of leading to risky or dangerous states.

[Li et al., 2018] shows another example of SafeRL where a constraining of the state space is used, in this article a RL framework that regulates exploration is used to ensure safety. A supervisor is present that evaluates the agent's chosen action and modifies it to ensure the output constraints are respected, the agent will then be penalized and eventually learn the admissible control sets (at which point the supervisor is removed).

The case for why constraining the state space is likely a better choice than including safety in the objective function as a way to ensure safe exploration is well developed in [Isele et al., 2018]. In this article a prediction model is used that works in parallel to the RL agent, this prediction model "masks" unsafe actions at each time step preventing the agent from choosing them. The proposed method achieves sub-optimal due to the fact that safety margins are used, that is, within a fixed time horizon each agent takes a single high level action, the variance acts as a bound for all the low level actions that produce similar high level actions to the one taken.

[Berkenkamp et al., 2016] uses the concept of Regions of Attraction (ROAs), regions where a control law can be given that will lead the system to a point of equilibrium, and extends it to allow for a safer exploration. The main contribution of the article is the fact that through their results they are able to account for the possibility of the estimated ROA having some overlap with the unsafe state space (due to uncertainties in the available model) by learning the ROA through experiments which allows it to stay within the true ROA thus achieving safety. If the system stays within an ROA it can safely explore since there will always be a control law that takes it back to a point of equilibrium (that is, obviously, safe in itself).

In [Luo et al., 2018] a system transformation is used (by resorting to tanh functions) that turns a constrained control problem (due to the fact there is limits to the outputs of the control surfaces) into an unconstrained control problem, thus it is an example of a restriction in state space to achieve safety. Value Iteration Q-Learning (VIQL) is then used to learn an optimal Q function and the weights of a critic NN are calculated. Ultimately these tools are used to design the adaptive constraint optimal control problem which uses gradient descent to search for solutions.

SafeRL through objective function

[Horie et al., 2019] shows one approach to including safety in the objective function, in this paper Multiple Objective Markov Decision Processes (MOMDPs), where there are several objectives to follow and thus the reward function is a vector function, were expanded to Multiple Objective Risky Markov Decision Processes (MORMDPs). Two algorithms to handle MOMDPs were presented: one (PrQ values) considers a probability of success when choosing each action and estimates that probability of success with SARSA exploration, the other one (EQ values) combines the previous one with Q learning by weighing the Q function with the probability of success. These methods were then applied to MORMDPs, this is because an MOMDP can be turned into an MORMDP by simply considering that one of the reward functions in the vector is a safety function.

Other miscellaneous articles

[Mirchevska et al., 2018] uses a different approach, a high-level system decides when/if to change lanes and a low level system executes the change. Safety verification is also implemented in combination with RL to assure safety.

In [Mannucci, 2017] a several-step-ahead prediction is used (with bounding models to provide the expected transitions) to choose the best safe action for the current state, a back up is also calculated (that corresponds to a series of actions that can take the system back to the neighborhood of a previously visited safe state). During each time step the agent will try to pick the best possible action, if none is found then it will automatically execute the backup and restart the process.

The methods that involve several-step-ahead predictions usually face a problem where, due to the fact the state transitions have to be propagated several times, the subset of possible arrival states grows increas-

ingly larger as the number of lookahead steps increases. In [Torsten Koller and Krause, 2018] this problem is addressed, this paper uses Gaussian Processes as well as ellipsoids to bound the state transitions. By doing a certain action the system can predict the next state will be within an ellipsoid, n-step ahead prediction is then possible. To ensure that the ellipsoids contract towards their center and don't just endlessly expand making the computation intractable (which allows for faster computations and further lookahead steps), affine state feedback laws are implemented to account for the possibility of the future inputs correcting for deviations of the model prediction. The series of feedback controllers obtained from the solution to the problem can then be used as a return strategy. A performance trajectory is incorporated within the problem along with the return strategy to guide the problem towards more optimal solutions, nevertheless in the first few time steps the policies have to be the same for both parts.

[Ma et al., 2018] uses a different method altogether than those discussed previously. This method is called Adversarial RL and it works by defining the control problem as a two player game, the first player is the **protagonist** whose job is to stabilize a car in a highway (for example) and the other player is the **adversary** that continuously provides disturbances as a way to destabilize the car (a cap is maintained on the maximum value of the disturbance that changes successively). The logic behind Adversarial RL is that if there's an agent that continuously tries to disrupt the protagonist's task then the protagonist will eventually learn and become resistant to that disruption. [Ma et al., 2018] focuses on comparing the results of two different types of Adversarial RL, that differ in the way they define the two player game:

- **Competitive** games are zero-sum games, in these games the goal of one agent is directly opposed to the other or, in RL terms, $r_2 = -r_1$.
- **Semi-Competitive** games are similar to competitive games except the adversary's reward now also has a component (r_c) that promotes cooperation, in RL terms: $r_2 = -r_1 + r_c$.

The paper concludes that strictly competitive games have some severe disadvantages such as: the protagonist becoming too cautious and stuck in local minima and the adversary becoming predictable (since it will always choose the greatest disturbance it is allowed to) which leads to overfitting. From the results achieve it became clear that a semi-competitive setting allows the protagonist to adjust better and achieve more satisfactory results in addition to making the adversary continuously adjust its distribution which, in turn, allows for a larger region of the disturbance space to be sampled.

3.3.3. SHERPA

As a proof of concept for the algorithms that will be studied and developed in the further stages of this research the SHERPA algorithm is briefly explained as an example of a SafeRL algorithm, in section 4.2 an example of the application of SHERPA will be shown. SHERPA was developed in [Mannucci, 2017] and [Mannucci et al., 2018].

The SHERPA flowchart can be seen in figure 3.6. The basic idea of SHERPA is that at each state we use a policy to generate a proposed action, that action is then propagated to predict where it ends up and if that state is safe, if it is safe the SHERPA algorithm will then try to find a backup (a sequence of actions that bring the agent back to the safe state space) if it can find it then it will transition to the proposed state, otherwise (after a set number of iterations) it will use the current backup and try again.

SHERPA's approach to Safe Reinforcement Learning is, then, to make sure that the actions taken not only lead the system to a new safe state but also that the agent has a backup (in the form of a sequence of actions) that will lead it back to the safe state space. Throughout the time SHERPA executes these operations the system relies on a warning function ($W(x)$) to determine whether the neighborhood of the current state is safe or not. This information from this warning function is also used to update what the Safe State Space is.

3.3.4. Conclusions

As stated in 3.3.2, state space constraining is likely to be a better choice when RL is done in an environment where safety is critical (aviation and self-driving cars, for example). This is mainly due to the fact that this approach limits the state space the agent can reach and thus it stops the agent from transitioning into safety-critical states where a failure may occur or that may lead to a state where a failure occurs. This approach, of course, leads to sub-optimal policies being derived which means that for tasks that do not necessarily have safety as a main goal this approach might not be the best one.

The approach of introducing safety as a criterion in the objective function should be avoided as even if the penalty for negative events is very high, if they happen with low probability then, due to averaging, it is policy that the agent will still choose those actions that transition into unsafe states.

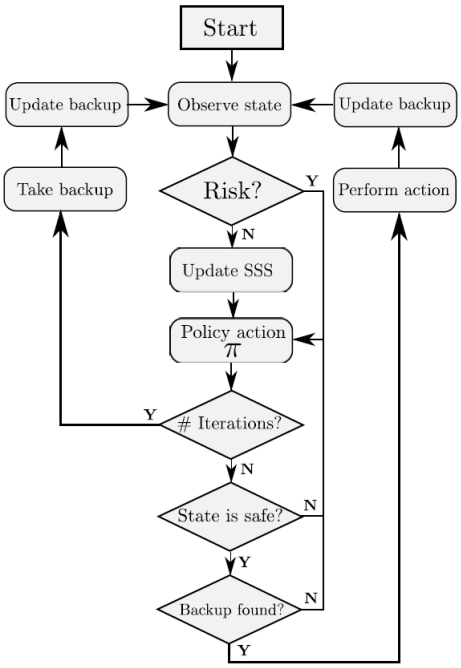


Figure 3.6: Flowchart for the SHERPA algorithm, taken from [Mannucci, 2017].

From the research conducted, it is quite clear that there is a substantial interest in the research field of Reinforcement Learning in coming up with an algorithm that can Safely explore and learn a policy without transitioning into safety critical states. It was seen during the research that a SafeRL algorithm could have an assortment of applications from self-driving cars to UAVs and full-sized commercial aircraft.

4

Preliminary Analysis

4.1. Application of the RL algorithms

This part of the report will be divided into three subsections. In the first one a few simpler examples of the RL algorithms will be discussed as ways of further describing their usage and results. In the second subsection the applications of the algorithms to more complex problems will be introduced. Finally, the third subsection will focus on one of the research questions and will investigate the performance of the operative and proximity metrics as standalone algorithms and a possible merging of both of them into a more convoluted one.

The overall goal of this part of the report is to show how RL can be applied in different circumstances and using different algorithms as well as showcasing some of the different characteristics of RL methods.

4.1.1. SARSA Cliff walking

The cliff walking is that described in 3.2.3 where we have an agent that has to get from a beginning position to and end goal position while trying to avoid falling into a cliff (since the penalty for falling there is extremely high). This task is the typical first example that is given for SARSA applications, not only because it is a good way to compare SARSA and Q-learning as was done in 3.2.3 but also because its a 2-D example that has a state space small enough for each state action value to be represented in memory discreetly. If we consider that the agent has 4 possible actions at each point (UP,DOWN,LEFT,RIGHT) and that there 4×12 squares in the gridworld, this means that our estimate of the state action value function (Q(s,a)) can be represented by a $4 \times 4 \times 12$ tensor, which is small enough to be stored and handled efficiently. For a more complex task with a much larger state space a more suitable representation of the Q(s,a) values would have to be used, such as a neural network or spline, since it would be inefficient to store each value as an entry in a tensor or table. One particular situation of this happens when we have a state space that is made up of continuous instead of discrete values, for example the flight envelope of an aircraft, in order to represent all the possible values of the state space we would need infinite memory.

The results for this example can be obtained by creating a table representing what the algorithm estimated to be the optimal policy at each state and by showing a heat map of the points visited throughout the experiment. The results shown in figure 4.1 and table 4.1 are obtained with the following parameters:

- $\epsilon = 0.1$
- $\gamma = 0.9$
- $\alpha = 0.5$
- 1000 episodes

[0] **This chapter has already been graded for AE4020.**

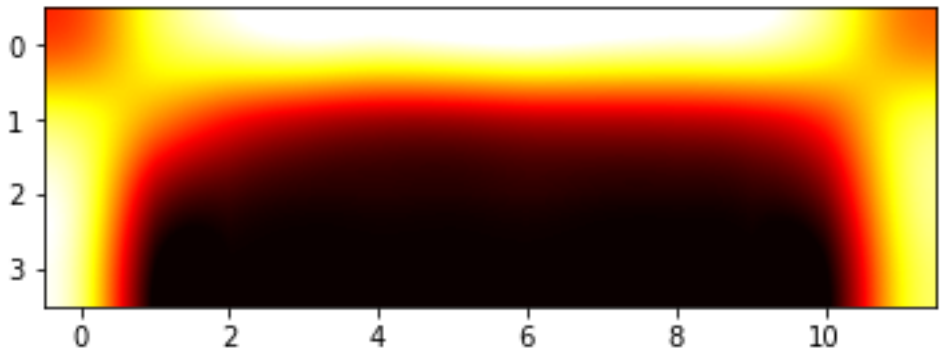


Figure 4.1: Heat Map for the SARSA cliff walking task.

Positions X/Y:	1	2	3	4	5	6	7	8	9	10	11	12
1	→	→	→	→	→	→	→	→	→	→	→	↓
2	↑	↑	↑	→	↑	↑	↑	→	→	↑	→	↓
3	↑	←	↑	←	↑	↑	↑	↑	↑	←	→	↓
4	↑	↑	↑	↑	↑	↑	↑	↑	↑	↑	↑	G

Table 4.1: Policy to which the SARSA algorithm converges

Note that in table 4.1 the Start position is the bottom left corner of the table and the end goal is the bottom right. What is shown by the heat map in figure 4.1 is the amount of times the algorithm visits each state (the darker the color, the less amount of times the algorithm visits a state). Overall we can see that the system avoids going to the very top of the table and also tries to avoid being close to the cliff (bottom line).

4.1.2. Q-Learning Cliff walking

This example is basically the same as in 4.1.1 except now we use a Q-learning algorithm instead of SARSA, this change mainly concerns the fact that now the Q-learning algorithm predictions and expectations for what the return from the next state can be (by following an $\epsilon - greedy$ policy) instead of actually taking an action and observing the return from that state.

The results shown in figure 4.2 and table 4.2 are given for the same parameters as in 4.1.1.

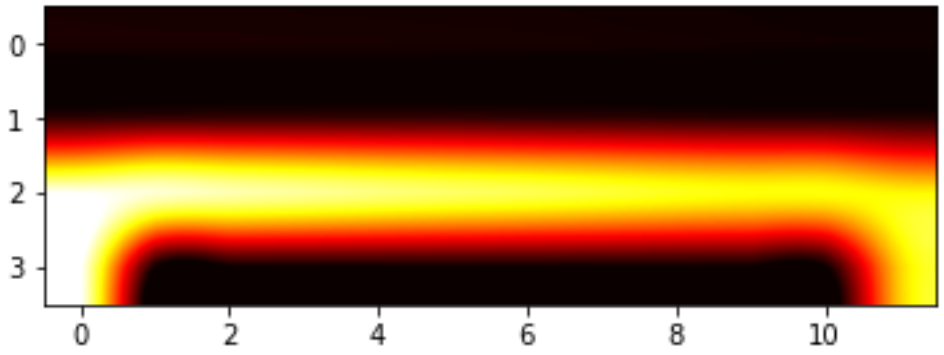


Figure 4.2: Heat Map for the Q-Learning cliff walking task.

Positions X/Y:	1	2	3	4	5	6	7	8	9	10	11	12
1	→	→	↑	→	↓	→	→	→	↓	↓	→	↓
2	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓
3	→	→	→	→	→	→	→	→	→	→	→	↓
4	↑	↑	↑	↑	↑	↑	↑	↑	↑	↑	↑	G

Table 4.2: Policy to which the Q-learning algorithm converges

Much like in 4.1.1, the heat map represents, again, the number of times each state is visited. The Start and Goal position remain the same as before.

From the heatmap given in figure 4.2 it is quite clear that the Q-learning algorithm does choose the optimal path as opposed to SARSA that usually chooses the safer path, as was discussed in 3.2.3. Table 4.2 shows very clearly that the algorithm converges to an optimal policy (which is to go to the goal while right alongside the cliff).

4.1.3. Q-learning windy gridworld

Another example that was tested was the windy gridworld, similar to the cliff gridworld except now in certain tiles there is a wind that "pushes" the agent upwards, as shown in figure 4.3 taken from [Sutton and Barto, 2017].

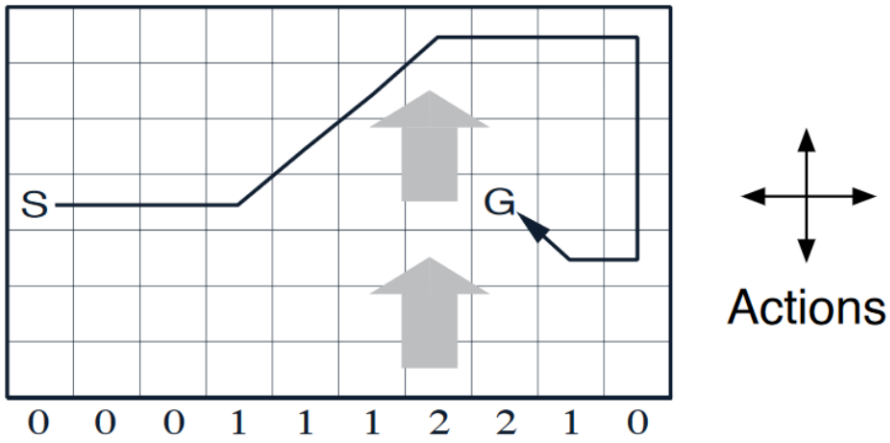


Figure 4.3: Windy gridworld setup

The parameters used in this case were similar to those with the cliff walking except the number of episodes was reduced to 300. The results are, as with the two previous examples, shown in figures 4.3 and 4.3. Due to the wind the lower right parts of the gridworld were never explored, they optimal policy is to go UP simply because they were never visited. It is quite clear that the strategy of the agent is to go all the way to the right side of the goal and then move left and use the wind to its advantage (By going to [6,10] and then [6,9] ending up in [4,8].

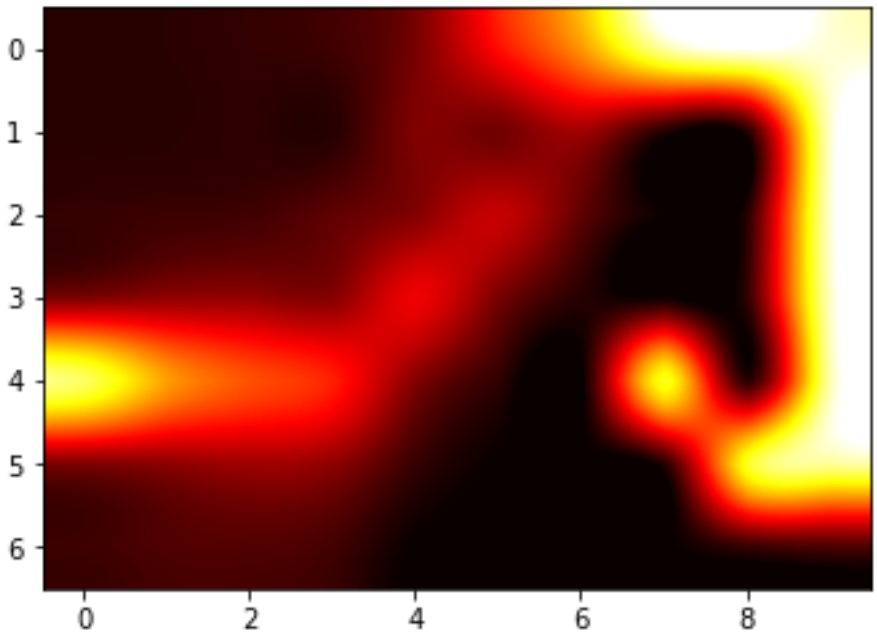


Figure 4.4: Heat Map for the Q-Learning windy gridworld task.

Positions X/Y:	1	2	3	4	5	6	7	8	9	10
1	→	→	→	→	→	→	→	→	→	↓
2	↑	→	→	→	→	→	↓	→	↑	↓
3	→	→	→	→	→	→	→	↑	→	↓
4	→	→	→	→	→	→	→	↑	→	↓
5	↓	→	→	→	→	→	↑	G	→	↓
6	→	→	→	→	↓	↑	↑	↓	←	←
7	→	→	→	↑	↑	↑	↑	↑	→	←
Wind Strength	0	0	0	1	1	1	2	2	1	0

Table 4.3: Windy gridworld with SARSA, converged policy

4.2. SHERPA

As was mentioned in 3.3.3, an example of the implementation of the SHERPA algorithm will now be shown as a proof of concept and demonstration of the usefulness of SafeRL algorithms

4.2.1. Proof of concept setup

The example used to show SHERPA's effectiveness is that of a quad rotor in a 2-D room navigating and exploring it. The room has a width of 3 meters and a height of 2 meters. This example was chosen since it is also reproduced in [Mannucci, 2017].

For this specific task the dynamics of the UAV quad rotor can be described through the following equations:

$$\ddot{x} = -\frac{[c]}{[m]}|sin(\theta)||\dot{x}|x + \frac{T}{[m]}sin(\theta); \tag{4.1}$$

$$\dot{\theta} = [k]l_{\tau}; \tag{4.2}$$

$$\ddot{z} = -\frac{[c]}{[m]}|cos(\theta)||\dot{z}|\dot{z} + \frac{T}{[m]}cos(\theta) - g \tag{4.3}$$

The state and input vectors are described by:

$$x = \begin{bmatrix} x & \dot{x} & z & \dot{z} & \theta \end{bmatrix}^T; \qquad u = \begin{bmatrix} T & l_\tau \end{bmatrix}^T \tag{4.4}$$

The pair (c, m) are bounded values that, in the case of the results generated for this examples were determined to be $m = 0.345$ and $c = 1.06 * 10^{-4}$. The initial state (x_0) was defined to be at the middle of the room $(x = 0, z = 1)$ with initial velocities of zero as well as initial $\theta = 0$.

The episode terminates when the maximum number of epochs is reached or the UAV hits the wall.

The policy to be followed is a random policy.

4.2.2. Results

Due to the fact the policy used is random, the results are always going to be limited in terms of how good they are. One of the things that was most noticeable when running the algorithm several times was that the main reason it failed to avoid hitting the wall was that in some runs the UAV would gather too much horizontal or vertical momentum and the UAV would be unable to recover due to the fact the control inputs are bounded. One of the way to ascertain the effectiveness of SHERPA is to see how long the UAV can 'survive' inside the room.

Results shown in table 4.4 are for 10 iterations for state look-up, 40 for backup look-up and 5 step ahead look up. Figure 4.5 shows a typical trajectory within one of these runs, in it the blue line represents the trajectory, the blue circles represent the states where a normal action was taken and the orange circles represent the states where the agent had to resort to using a backup. Note that the orange circles are further apart than the blue ones because a backup is a series of actions whereas, during normal operation, the agent only takes one action at a time. In figure 4.5 the number of normal actions taken was 291 and the number of times the system had to resort to a backup was 9 (as can be easily seen by the number of orange circles).

It can be seen that SHERPA has an average length that is ≈ 15% higher than the random policy without SHERPA, this means the SHERPA algorithm increases the survivability of the UAV even when the policy is fully randomized.

Method	Total Number of epochs	Average duration of episode
SHERPA	1715256	171.5256
Pure Random	1480288	148.0288

Table 4.4: Results on the survivability of the UAV

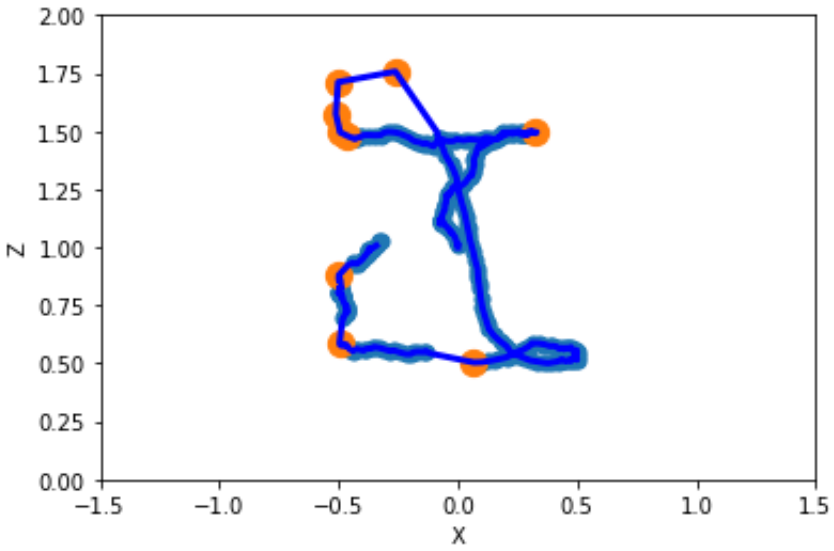


Figure 4.5: A typical trajectory completed by the algorithm

The results were found to be very dependent on how many steps ahead the algorithm looked which made

sense since, the further it looks ahead, the more likely it is to be able to counter negative trends early enough so that it won't hit the wall.

By simply doubling the number of steps ahead the algorithm looks from 5 to 10 the number of total epochs increased to 2159814 (which means a 215.9814 average duration of each episode), which means the algorithm then performs 45% better than purely random, which is a large improvement.

This example goes to show that even when the policy is not adequate (random policy in this case) SHERPA is very effective at increasing the survivability of the UAV during its trials. These results are consistent with the ones presented in [Mannucci, 2017].

One of the main issues with using an increasing number of steps ahead for prediction is that the computation time also increases fairly quickly as the number of steps-ahead goes up. This will be one of the challenges that will likely be addressed during the next stages of the research.

5

Additional Results

5.1. Calculating variance and mean for the Gaussian Process

When using the ellipsoid bounding model, a Gaussian Process (GP) is used to model the model error. This GP is defined by a mean, μ, and a variance, σ. In order to correctly use the ellipsoid model, therefore, values have to be found for these parameters so that the propagation of the ellipsoids can be conducted.

To obtain these values, as was done in [Torsten Koller and Krause, 2018], the system was placed into the initial state:

$$x = x_0 = \begin{bmatrix} 0 & 0 & 0 & 0 \end{bmatrix}^T \tag{5.1}$$

From this initial state, an action was performed and the difference between the predicted state ($h(x_t, u_t)$) and the actual state ($f(x_t, u_t)$) at the next time step was calculated. This difference ($g(x_t, u_t)$) is the one to be modeled by the GP, therefore, the measurements of this error can be used to calculate the mean and variance of the GP.

In order to obtain a more accurate measurement of this values this process was repeated a number of times. The system was initialized at x_0, then a random action from the action set was selected and the error was calculated. After n such iterations of this process the covariance and mean of the error was calculated and used as that of the Gaussian Process. This process is repeated for each initialization of the algorithm whenever the actual model is selected randomly, such as is the case of the elevator control task.

The parameter n was set to 50. The reason for the selection of this value can be seen in figures 5.1 and 5.2. In these figures it can be seen that increasing the number of iterations used to calculate the covariance does not significantly alter its value, this is to be expected since the action set available to the agent is limited. Therefore a value of $n = 50$ was found to be acceptable at finding values for the covariance and mean without significantly increasing the computation time.

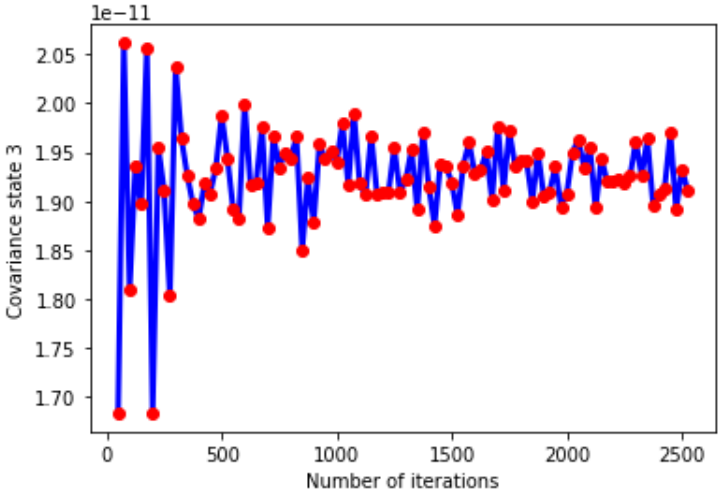


Figure 5.1: Covariance values obtained for the α state.

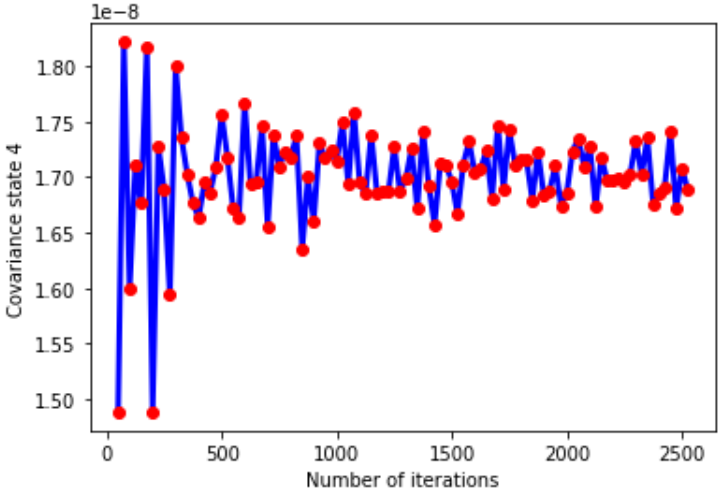


Figure 5.2: Covariance values obtained for the q state.

6

Conclusions and future work

6.1. Answering the research questions

The research questions posed in section 1.4 can now be reiterated and answered. Some had already been answered through the literature review and some required simulations to be run in order to obtain results to validate their response.

The research questions and their respective answers are as follows:

1. How can Reinforcement Learning be used in control tasks for safety-critical systems?

   A: If a Safe RL approach is taken, where safety is taken as one of the primary objectives of the learning algorithm, then an agent can be used in these kinds of control tasks.

2. What is the current state of the art in Safe Reinforcement Learning

   A: This question was answered in 3.3.2, it was seen that most of the current work on Safe Reinforcement Learning focuses on either constraining the state space to a subset that is guaranteed to be safe or in including safety as a part of the objective function. In [García and Fernández, 2015] a good description for the state of the art is also included however it does not include contributions beyond its year of publication (2015).

3. Of the two main approaches to Safe Reinforcement Learning which one is more adequate?

   A: The most adequate approach is to constrain the state space to a subset that includes only states that are known to be safe. This guarantees that the system does not transition into safety-critical states. If safety is included into the objective function then there is always a chance that, due to averaging, even giving very high penalties to critical states might not be enough because their return might not be as low as one would want causing those states to be visited.

4. How can an algorithm be designed that achieved Safe Reinforcement Learning?

   - How can the operative and proximity metrics be combined in order to achieve a SafeRL algorithm?

     A: The operative and proximity metrics were combined through a method that uses one for regular exploration (proximity) and the other for recovery whenever danger is detected and a warning appears (operator). This method essentially corresponds to using a new metric altogether, named ProxOp.

   - How can the ellipsoids discussed be used as a way to reduce model uncertainty?

     A: The ellipsoids can be used as an alternative to the interval analysis method and can indeed provide some benefits when it comes to dealing with model uncertainty, namely when information about the bounds of the system model is either inaccurate or unavailable.

     Ultimately, combining the ProxOp metric with the ellipsoid bounding model is a viable alternative to achieve Safe Reinforcement Learning exploration and solve some of the problems discussed in section 1.2.

6.2. Future work

The focus of this thesis was upon advancing the state of the art towards a SafeRL solution that could, one day, be used as a full autopilot system for both conventional aircraft and UAVs. A viable combination of the two metrics was developed and an alternative bounding model with better characteristics towards dealing with uncertain systems was found and applied.

Although the results achieved were satisfactory, there are still some challenges to be overcome before the research carried out here can be used in the commercial market. These challenges are mostly focused around two aspects: **Safety** and **Online Efficiency**.

Firstly, when it comes to **Safety** it is clear that for a control method to be deployed on a commercial aircraft it needs to have a stellar track-record when it comes to safety. Although the algorithms presented here have high success rates, these success rates might still not be enough for the control system to be certifiable. Further work is needed when it comes to guaranteeing additional safety and better performance. One possible way to increase safety would be to have look ahead several more time-steps using the state propagation to verify if, by looking further into the future, the agent would be able to detect danger earlier on and, thus, have a greater success rate. Increasing the number of time-steps one looks ahead, however, increases the computational load on the system since more calculations have to be done at each time-step to determine the best action.

Secondly, **Online Efficiency** is a major obstacle for learning algorithms in general, and for some of the algorithms presented here in particular. The proximity metric is very computationally efficient and has a run time that enables it to run in real time. Both the operative and ProxOp metrics, however, are computationally demanding and take a considerable amount of run time to finish each iteration making them unable to perform in real time in their current state. Further work is needed in this area, one appealing prospect would be to find a way to update the values of the tiles in a more efficient manner or to find an alternative way to store and update information for the value of the tiles. If these changes were made these metrics could be usable for real time applications.

Another possible point for expansion on the research developed here has to do with the systems in which these algorithms have been tested. Since the ellipsoid bounding model is able to deal with both unknown model errors, process noise and sensor noise it would be interesting to expand the algorithm to systems where these factors are significant in their operation.

Bibliography

B. D. Argall, S. Chernova, M. Veloso, and B. Browning. A survey of robot learning from demonstration. *Robotics and Autonomous Systems*, 57(5):469–483, 2009.

E. Barnard. Temporal-difference methods and markov models. *IEEE Transactions on Systems, Man and Cybernetics*, 23(2):357–365, 1993.

F. Berkenkamp, R. Moriconi, A. P. Schoellig, and A. Krause. Safe learning of regions of attraction for uncertain, nonlinear systems with gaussian processes. In *2016 IEEE 55th Conference on Decision and Control, CDC 2016*, pages 4661–4666, 2016.

K. Driessens and S. Džeroski. Integrating guidance into relational reinforcement learning. *Machine Learning*, 57(3):271–304, 2004.

Z. Fang, C. Hao, and P. Li. Learning linear parameter-varying control of small-scale helicopter using episodic natural actor-critic method. In *2012 12th International Conference on Control, Automation, Robotics and Vision, ICARCV 2012*, pages 1001–1005, 2012.

S. Ferrari and R. F. Stengel. An adaptive critic global controller. *Proceedings of the American Control Conference*, 4:2665–2670, 2002.

J. García and F. Fernández. A comprehensive survey on safe reinforcement learning. *Journal of Machine Learning Research*, 16:1437–1480, 2015.

B. Gaudet and R. Furfaro. Adaptive pinpoint and fuel efficient mars landing using reinforcement learning. *IEEE/CAA Journal of Automatica Sinica*, 1(4):397–411, 2014.

A. Hans, D. Schneegaß, A. M. Schäfer, and S. Udluft. Safe exploration for reinforcement learning. In *ESANN 2008 Proceedings, 16th European Symposium on Artificial Neural Networks - Advances in Computational Intelligence and Learning*, pages 143–148, 2008.

S. Harm van, H. Hado van, S. Whiteson, and M. Wiering. A theoretical and empirical analysis of expected sarsa. In *2009 IEEE Symposium on Adaptive Dynamic Programming and Reinforcement Learning, ADPRL 2009 - Proceedings*, pages 177–184, 2009.

Matthias Heger. Consideration of risk in reinforcement learning. In William W. Cohen and Haym Hirsh, editors, *Machine Learning Proceedings 1994*, pages 105 – 111. Morgan Kaufmann, San Francisco (CA), 1994. ISBN 978-1-55860-335-6. doi: https://doi.org/10.1016/B978-1-55860-335-6.50021-0. URL `http://www.sciencedirect.com/science/article/pii/B9781558603356500210`.

N. Horie, T. Matsui, K. Moriyama, A. Mutoh, and N. Inuzuka. Multi-objective safe reinforcement learning: The relationship between multi-objective reinforcement learning and safe reinforcement learning. *Artificial Life and Robotics*, 2019. Article in Press.

David Isele, Alireza Nakhaei, and Kikuo Fujimura. Safe reinforcement learning on autonomous vehicles. In *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS 2018, Madrid, Spain, October 1-5, 2018*, pages 1–6, 2018. doi: 10.1109/IROS.2018.8593420. URL `https://doi.org/10.1109/IROS.2018.8593420`.

D. J. Lee and H. Bang. Reinforcement learning based neuro-control systems for an unmanned helicopter. In *ICCAS 2010 - International Conference on Control, Automation and Systems*, pages 2537–2540, 2010.

Z. Li, U. Kalabić, and T. Chu. Safe reinforcement learning: Learning with supervision using a constraint-admissible set. In *Proceedings of the American Control Conference*, volume 2018-June, pages 6390–6395, 2018.

W. S. Lovejoy. A survey of algorithmic methods for partially observed markov decision processes. *Annals of Operations Research*, 28(1):47–65, 1991. Cited By :286.

B. Luo, D. Liu, and H. . Wu. Adaptive constrained optimal control design for data-based nonlinear discrete-time systems with critic-only structure. *IEEE Transactions on Neural Networks and Learning Systems*, 29 (6):2099–2111, 2018.

X. Ma, K. Driggs-Campbell, and M. J. Kochenderfer. Improved robustness and safety for autonomous vehicle control with adversarial reinforcement learning. In *IEEE Intelligent Vehicles Symposium, Proceedings*, volume 2018-June, pages 1665–1671, 2018.

T. Mannucci, E. . Van Kampen, C. De Visser, and Q. Chu. Safe exploration algorithms for reinforcement learning controllers. *IEEE Transactions on Neural Networks and Learning Systems*, 29(4):1069–1081, 2018.

Tommaso Mannucci. *Safe Online Robust Exploration for Reinforcement Learning Control of Unmanned Aerial Vehicles*. PhD thesis, Delft University of Technology, 2017.

D. Martínez, G. Alenyà, and C. Torras. Safe robot execution in model-based reinforcement learning. In *IEEE International Conference on Intelligent Robots and Systems*, volume 2015-December, pages 6422–6427, 2015.

B. Mirchevska, C. Pek, M. Werling, M. Althoff, and J. Boedecker. High-level decision making for safe and reasonable autonomous lane changing using reinforcement learning. In *IEEE Conference on Intelligent Transportation Systems, Proceedings, ITSC*, volume 2018-November, pages 2156–2162, 2018.

International Civil Aviation Organization. *Universal Safety Oversight Audit Programme Continuous Monitoring Manual*. International Civil Aviation Aviation Organization, Montréal, Quebec, Canada, 3 edition, 2011.

J. Pearl. Heuristics: Intelligent search strategies for computer problem solving. 1 1984.

Martin Pecka, Karel Zimmermann, and Tomas Svoboda. Safe exploration for reinforcement learning in real unstructured environments. In *20thComputer Vision Winter Workshop, Seggau, Austria, February 9-11, 2015*, 2015.

R. S. Sutton, D. Precup, and S. Singh. Between mdps and semi-mdps: A framework for temporal abstraction in reinforcement learning. *Artificial Intelligence*, 112(1):181–211, 1999.

Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning:An Introduction*. The MIT Press, Cambridge, Massachusetts, 2 edition, 11 2017.

Matteo Turchetta Torsten Koller, Felix Berkenkamp and Andreas Krause. Learning-based model predictive control for safe exploration. *IEEE Conference on Decision and Control*, 2018. Article in Press.

E. Van Kampen, Q. P. Chu, and J. A. Mulder. Online adaptive critic flight control using approximated plant dynamics. In *Proceedings of the 2006 International Conference on Machine Learning and Cybernetics*, volume 2006, pages 256–261, 2006.

C. J. C. H. Watkins and P. Dayan. Technical note: Q-learning. *Machine Learning*, 8(3):279–292, 1992.

H. Zhang, S. Wei, and G. Xu. Steady state controller design for aero-engine based on reinforcement learning nns. In *Proceedings of the 29th Chinese Control and Decision Conference, CCDC 2017*, pages 2168–2173, 2017.