Reinforcement Learning based Algorithm with Safety Handling and Risk Perception

Shyamsundar, S.; Mannucci, Tommaso; van Kampen, Erik-Jan

DOI
10.1109/SSCI.2016.7849367

Publication date
2016

Document Version
Accepted author manuscript

Published in
2016 IEEE Symposium Series on Computational Intelligence

Reinforcement Learning based Algorithm with Safety Handling and Risk Perception

Suhas Shyamsundar, Tommaso Mannucci, Erik-Jan van Kampen

Faculty of Aerospace Engineering, Delft University of Technology, Delft, 2629 HS, the Netherlands

***Abstract*—Navigation in an unknown or uncertain environment is a challenging task for an autonomous agent. The agent is expected to behave independently and to learn the suitable action to take for a given situation. Reinforcement Learning could be used to help the agent adapt to an unknown environment and learn the right actions to take. This paper presents the setup and the results of a reinforcement learning problem utilizing Q-learning and a Safety Handling Exploration with Risk Perception Algorithm (SHERPA) for safe exploration in an unknown environment. The agent has to explore its environment safely and must learn the optimal action for a given situation from the feedback received from the environment. The results show that the agent can learn a value function converged to within 10% of the optimal values after 5000 iterations. The simulation results show that the proposed approach ensures that the agent explores an unknown environment safely and learns the desirable actions for a given situation.**

I. Introduction

Reinforcement learning [1] (RL) is a bio-inspired control approach based on agent-environment interaction. While traditional machine learning algorithms are susceptible to dynamic changes in the environment, RL can help an autonomous agent learn the optimal behaviour, with little or no prior knowledge of its environment, and with changing conditions. For these reasons, RL has become a promising tool for improving autonomous navigation of an agent in an unknown environment.

During exploration, the agent tries different actions to learn and refine its policy. However, actions that put the agent in danger are undesirable, e.g. on collision course with an obstacle. For profitable exploitation in real-life applications, it is necessary that the agent explores safely. Existing RL algorithms tend to encourage exploration, but only a few take into account the repercussions of exploration in a potentially hostile environment. Fraichard and Asama [2] propose an algorithm that ensures safety in the exploration phase of an agent's learning to a certain extent. A drawback is that the algorithm relies on a-priori knowledge of the environment. Moldovan and Abbeel [3] propose an optimization formulation algorithm that guarantees safe exploration, at the cost of sub-optimal exploration. This algorithm has been shown to perform better exploration than classical exploration methods, frequent updates make it computationally expensive. Inspired from [2] and [3] is the Safety Handling Exploration with Risk Perception Algorithm (SHERPA) [4]. Developed to comply with model-free algorithms, SHERPA enforces safety of exploration by searching the current set of actions for which the safety of undertaking is proven earlier, with limited risk perception by the agent. A drawback is once again the large computation time required by the algorithm which makes it unfeasible for online applications. Polo and Rebollo [5] propose a RL algorithm that uses a previously defined safe policy which is assumed to be sub-optimal, to learn a policy that is near optimal. B. Zuo et al. [6] propose a Q-learning based navigation algorithm for autonomous systems that shows good performance and learning for safe navigation in an unknown environment. The concept of Q-learning has been used to trace an optimal path for navigation in grid-based environments [7]. The Q-learning approach is found to be effective for autonomous navigation in an unknown environment with relatively good real-time performance and reduced complexity [6], [7].

The goal of this paper is to provide a RL algorithm using Q-learning for navigating an unknown environment, with a focus on on-line performance. To ensure safety of the agent during the learning phase, SHERPA is used to determine the safety of a proposed action for a given situation, while on-line performance is addressed by reducing the dimensionality of the learning problem to a minimum. The inclusion of SHERPA results in a safe exploration algorithm, which performs entirely on-line avoiding off-line training and with limited a-priori knowledge of the agent dynamics, and which constitutes the main contribution of this work.

The rest of the paper is structured as follows. Section II presents the fundamentals of RL, and introduces Q-learning and SHERPA. Section III, presents the learning environment and quadrotor agent used to test the approach. Section IV presents two algorithms: a base Q-learning algorithm and the same algorithm augmented with SHERPA. Results of the application of the algorithms are included. Section V compares the two algorithms in light of the results. Finally, section VI concludes.

II. Fundamentals

In RL, learning comes from direct interaction with the environment modeled as a *Markov Decision Process* (MDP): at each time t the agents selects action a_t, observes a transition from previous state s_t to the next state s_{t+1}, and receives a reward feedback r_t. The reward determines the intrinsic desirability of the transition, what is good in an immediate sense. Rewards and transitions are used to update the policy π, which defines the agent's behaviour by mapping each state to an action. Given the policy and the discount β, the value

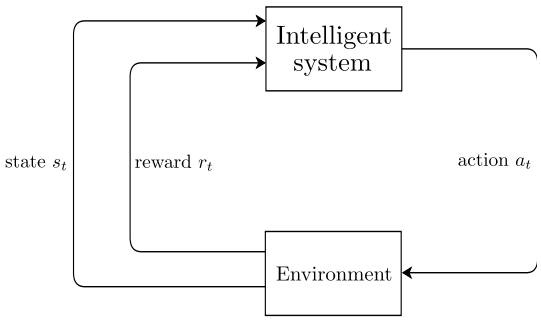


Fig. 1: Agent-Environment Interface [1]

function, $V^{\pi}(s)$ represents the value of states, i.e. the sum of discounted reward attainable from each state. The value function specifies what is good in the long run, the long term desirability of states after taking into account those that are likely to follow given the policy.

A. Q-learning

Q-learning is an off-policy, model-free RL algorithm [8], [9], i.e. it can learn the optimal policy, irrespective of the policy the agent is following and without a model of the MDP, as long as there is no bound that is placed on the number of times an action is executed. In its simplest form, Q-learning can be represented by one-step Q-learning, defined by

$$Q(s,a) \leftarrow Q(s,a) + \eta\left(r + \beta \max_a Q(s',a) - Q(s,a)\right) \quad (1)$$

where Q is the action-value function, $0 \leq \beta \leq 1$ is the discount factor and η is the learning rate. Q indicates the sum of discounted return that can be obtained when selecting action a in state s, so that he optimal policy consists of selecting the action with the highest value in each state.

B. SHERPA

SHERPA [4] defines a new control strategy that stems from a generalization of already known techniques (collision avoidance and reachability analysis) dedicated to the exploratory problem. SHERPA makes a distinction between the Fatal State Space (FSS) and the Safe State Space (SSS), and defines a new category of states; the *lead-to-fatal* (LTF) states. LTF states are those that are not fatal but, when encountered, will lead the agent to end up in the FSS with probability of one. The main goal of SHERPA is then to avoid these states. SHERPA, adopts an intermediate strategy between model-based and model-free learning: a *bounding model* of the agent is used to to estimate and bound the evolution of the agent in near time, but not to perform value iteration. Depending on the estimates, SHERPA promotes those actions that are more likely to provide safety of exploration.

III. Problem Setup

This section presents the simulation environment, and the RL algorithm designed to allow the agent, representing a quadrotor, to safely explore such environment. The quadrotor model is presented as well.

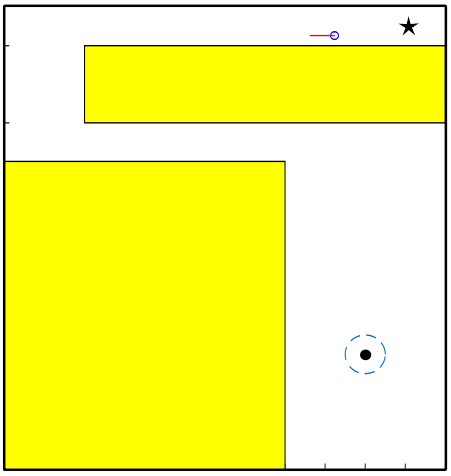


Fig. 2: The simulation environment is 12m x 12m in area. The circle, with a line indicating the heading, represents the agent, which is initialized at the position indicated by the star. The grey blocks surrounding the environment are obstacles. The black dot represents the goal. When the agent reaches within 0.1m of the goal, it is reinitialized in the upper corridor.

Figure 2 shows the simulation environment to evaluate the Q-learning algorithm: a corridors ending in a room where the goal is located. The task of the quadrotor agent is simply to reach the goal, while navigating safely, i.e. without colliding with the boundaries of the environment. Each episode starts with the agent in the upper corridor of the experiment, and ends after of 50 timesteps, during which the agent will try to reach a goal position in the room at the end of the corridor. Whenever the agent is successful in doing so, it is randomly reinitialized in the upper corridor.

The non-linear, two dimensional bounding model of the quadrotor is described by

$$\ddot{x} = -\frac{[c]}{[m]}|\dot{x}|\,\dot{x}\,cos\theta + \frac{T}{[m]}cos\theta \quad (2)$$

$$\ddot{y} = -\frac{[c]}{[m]}|\dot{y}|\,\dot{y}\,sin\theta + \frac{T}{[m]}sin\theta \quad (3)$$

where $[c]$ is the interval of damping coefficient c and $[m]$ is the interval of mass m:

$$[m] = [0.3440.516]kg \quad [c] = [0.721.08] \times 10^{-5}\frac{kg}{m}$$

T is the thrust generated by the rotors whose direction of application is specified byθ. The direction of the applied thrust is decided by the action to be executed. An action that suggests the agent to turn right will induce a thrust at an angle

perpendicular (90^0 or 270^0) to the direction the agent is facing. The thrust used to control the quadrotor is of 16N, equal to 80% of the total thrust $T_{\max}$=20N. To compute the real dynamics of the quadrotor, the 'true' model was obtained from the bounding model by selecting for each of the parameters its mean value.

The direction of motion θ is selected according to the 3 actions defined below:

- Action L: the agent turns left by 90^0 and move forward.
- Action R: the agent turns right by 90^0 and move forward.
- Action F: the agent moves forward in the direction it is facing.

The set of available actions can thus be defined as $a \in \{Left, Right, Front\}$. If the agent collides with an obstacle or the boundaries, it bounces backwards of 5cm.

IV. Algorithm

This section presents two RL algorithms for safe exploration and their results when applied to the quadrotor task of the previous section. The first algorithm presented uses pure Q-learning. Safety is addressed by implementing a direct reward that will teach the agent which actions is the safest. In order to accelerate learning, a simplified state representation based on distances from obstacles is adopted. The second algorithm augments previous control with SHERPA, which evaluates the safety of actions through interval estimation of the dynamics, and provides backups options to the agent.

A. Reward

Two different types of rewards are used in this paper, with the double objective of enforcing safety and reaching the goal. *Direct* reward is assigned depending only according to state-action pairs. For example, in figure 4, the agent is in the state $s = (123)$. The safest action that can be executed in this state is the action of moving forward (action *Front*). If the agent moves forward, it receives then a relatively large reward (+5), while turning left, which corresponds to the least safe option, will give it the least reward (+0). The intermediate action of turning *Right*, yields a moderate reward of +1. By means of direct reward, the agent can learn to avoid collisions without any actually occurring. If nonetheless a collision does occur, an *indirect* reward of -7 is assigned to further reinforce safe behaviour. Finally, a +10 reward is assigned when the goal is reached. The reward assigned is then $r{=}r_d + r_{id} + r_g$.

B. State representation

In order to reduce the otherwise large number of states-action pairs, a local state inspired from [14] is proposed that represent the agent's perception of its surroundings. The states are defined on the basis of the distance of the agent from the obstacles. It is assumed that 3 directional sensors are mounted on the agent, providing readings of the distance to the obstacles in 3 specific directions: front (F), left (L) and right (R). Additionally, a proximity sensor. A schematic representation of the agent and of its sensors is shown in figure 3.

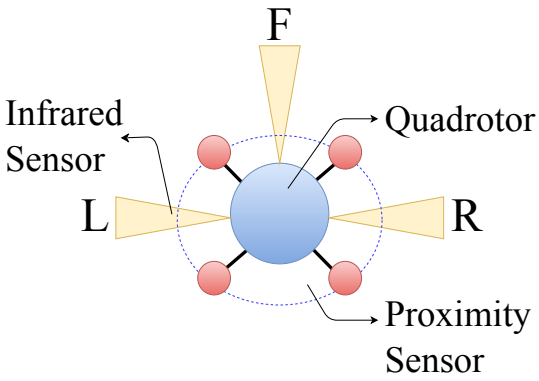


Fig. 3: Schematic representation of sensors.

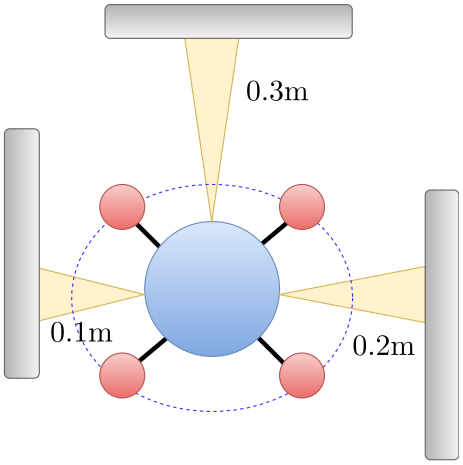


Fig. 4: A sensor measurement example.

On the basis of the sensor readings, a 3 element world vector $s = (LRF)^T$ is created, whose components order the distances of the agent from the obstacle in the three directions left, right and front. The direction whose distance from the nearest obstacle is the greatest is represented by integer '3'; the direction closer to obstacles is represented by '1'; the remaining distance is indicated by '2'. Figure 4 shows an example where the state vector is given by $s = (1, 2, 3)$: closest to obstacles on left distance, furthest at the front, and in between the two distances at right. By discretization of the states of the environment, it is possible to reduce the otherwise very large environment into only eight states.

C. Q-learning Algorithm

Figure 5 shows the Q-learning controller and the different types of reward. As mentioned earlier, the reward is shaped as to promote actions that are less likely to cause collisions; therefore, as time increases, the higher the value of an action state pair, the safest it is. In order to take advantage of this, the following modified ϵ-greedy policy is implemented:

- An initial value ϵ=0.7 is assigned for the first episode;
- with probability ϵ, a random action is selected; otherwise,

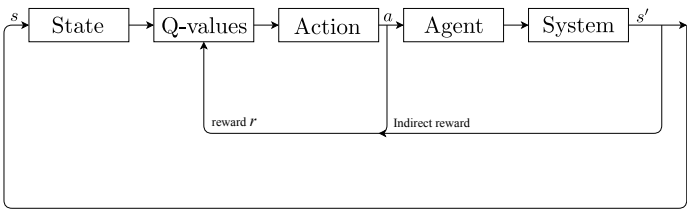


Fig. 5: Controller Architecture with Q-learning.

action a_k is taken with *softmax* probability

$$P(a_k) = \frac{e^{Q(s,a_k)/\tau}}{\sum_{l=1}^{m} e^{Q(s,a_l)/\tau}}$$

where temperature τ is updated every episode as $\tau \leftarrow \tau \cdot 0.9$ down to a minimum of 0.001;

- at the start of episode 10, 20 and 30, ϵ is updated as $\epsilon \leftarrow \epsilon - 0.2$;
- starting from episode 40, actions are selected greedily.

The above policy is used to dynamically balance exploration and exploitation. In the unknown environment, exploration is necessary; however, the softmax approach increases exploitation and therefore safety with time. The softmax approach uses a temperature variable τ. Higher τ cause the actions to be almost equiprobable, while lower τ increase the chance of selecting the action with the highest value. Eventually, a greedy action selection is adopted.

1) Results of the Q-learning Algorithm: The results of the Q-learning algorithm for a series of 100 episodes are presented. The algorithm is tested in the simulation environment of figure 2 assigning the reward as above. The results are given in table I for different combinations of learning rate η and discount factor β. It can be observed that the algorithm does not entirely prevent collisions. This is due to the fact that, even though the reward is shaped as to do so, the agent selects a significant number of random actions in the first 40 episodes. A greedier policy might reduce the number of collisions but, in general, relying on reward alone does not guarantee safety.

TABLE I: Comparison of results of the Q-learning algorithm for different values of η and β run for 100 episodes

	η=0.3, β=0.5	η=0.5, β=0.7	η=0.9, β=0.9
Random actions	877	848	830
Greedy actions	4123	4152	4170
Collisions	146	139	126
Goal reached	28	32	27
Total reward	21120	20980	20971

The following consideration must be made for the learning rate η. Adopting the relative representation of the environment $s = (LRF)^T$ means that distinct absolute states are represented by the same vector state. Figure 6 shows how increasing η leads to fluctuations when updating the action-value function $Q(s,a)$. Therefore an intermediate value η=0.5 is chosen for this application.

Consider again figure 6. Regardless of the learning rate adopted, the agent learns the safest policy. E.g., since obstacles

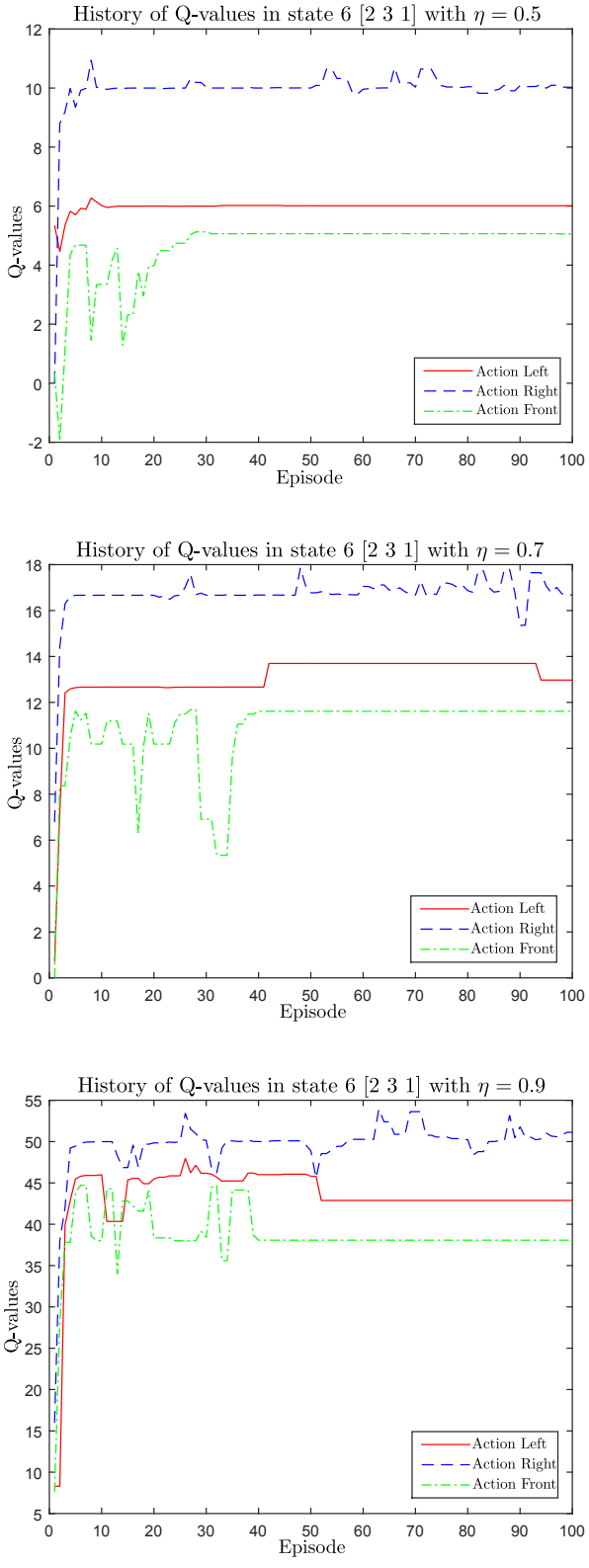


Fig. 6: History of Q-values of a state-action pair with three different learning rates η for state s=(2,3,1). For higher rates, more marked fluctuations are observed.

are the most distant at the right (indicated by the integer 3), action "right" is the safest, which is reflected in the Q-function. Nonetheless, a significant number of unsafe actions needs to be explored before this result can be achieved, which lead to collisions as reported in table I. In the real world, such collisions can be fatal for the agent and should be avoided.

D. Q-learning with SHERPA

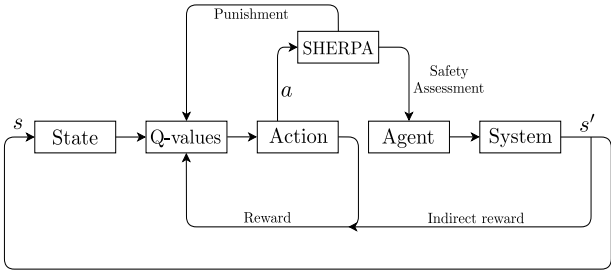


Fig. 7: Controller Architecture with Q-learning and SHERPA

SHERPA can be added to the previous algorithm to increase safety of the agent during the learning stage: figure 7 shows the augmented controller architecture. SHERPA works in *exploration* mode, and in *backup* mode. In the exploration mode, SHERPA evaluates the safety of the proposed action. In the backup mode, backups are generated for the proposed action.

In exploration mode, SHERPA verifies the action a_p proposed by the Q-learning algorithm for safety in the exploration mode. The boundaries of next state $[x_p]$ are predicted using interval analysis, and used to perform a two safety check. The first check consists of verifying if the interval $[x_p]$ is within the safe area of the environment. The history of sensor readings is consulted to ensure the quadrotor will not collide with the walls at its next position. For example, if the nearest obstacle is detected to be at 0.5m, and the computed interval $[x_p]$ is within this distance, the action is safe. If this check fails, SHERPA assigns a negative reward equal to -5 to the agent. Then, it proposes a new action from the set of available actions and evaluates its safety. SHERPA switches then to backup mode.

If action a_p passes the first check, a second one is made for the existence of a backup from predicted state $[x_p]$. A backup is essentially a safe option in the event none of the normal actions $a \in \{Left, Right, Front\}$ are accepted by SHERPA when in x_p. In this mode, the action set of the agent is augmented as follows. First, the quadrotor has the possibility to move backwards, so that $a_b \in \{Left, Right, Front, Back\}$. Second, a different amount of thrust between 0 and T_{max} can be selected.

Backups are checked as follows. A random chain of actions of variable length is generated. Starting from predicted state $[x_p]$, the bounding model provided by (2) and (3) is used to predict the boundaries of the state during the application of the actions. Sensor information is checked again to ensure that the agent keeps at distance from the obstacles. Multiple backups are tested, until a fixed number of iterations is reached. If a valid backup is not found by then, SHERPA discards action a_p, assigns negative reward, and proposes a different initial action.

1) Results of Q-learning with SHERPA: The Q-learning algorithm augmented with SHERPA is tested during 100 episodes, with $\beta = 0.7$ and $\eta = 0.5$. Figure 8a shows the corrective negative reward issued by SHERPA during these episodes. It is considerably high in the exploring phase of the agent, i.e. the first 40 episodes. This is to be expected, as random actions are often selected. The punishment decreases with time as the policy becomes more and more greedy; nonetheless, negative rewards are issued even during the exploitation phase. This means that in some iterations, SHERPA will prevent the action suggest by the Q-value function, even when this is the "optimal" action. Figure 8 shows the effect of the corrective reward on Q-values. Compared to the base algorithm, Q-values do not approximately converge when using the same learning rate. However, the Q-values for the best actions in the particular states are still higher than the others, so that although the values themselves do not converge, the optimal policy is still found.

The main benefit of implementing SHERPA can be appreciated from table II: the agent with the augmented algorithm does not incur in any collision. This means that SHERPA is able to keep the agent safe during the exploration phase of the algorithm. It can be seen that the agent has executed a backup action 40 times. If required, SHERPA is able to find at least one backup action for the agent.

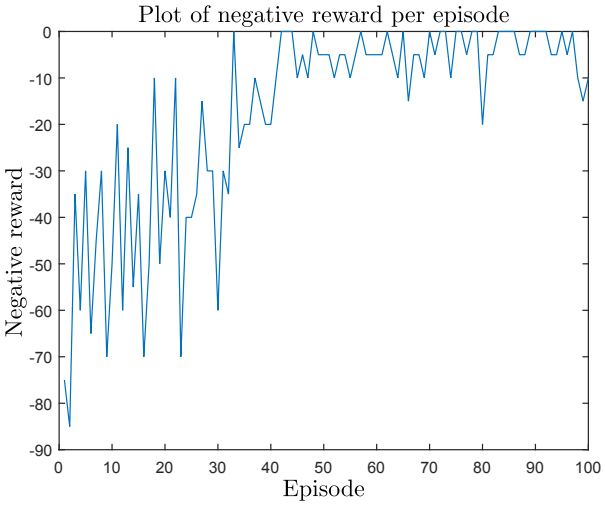
Parameter	Value
Number of collisions	0
Number of actions corrected	365
Number of times a safe action was not found	40
Number of times a backup was not found	0

TABLE II: Results of the Q-learning algorithm with SHERPA for η=0.5 and β=0.7 for a run of 100 episodes

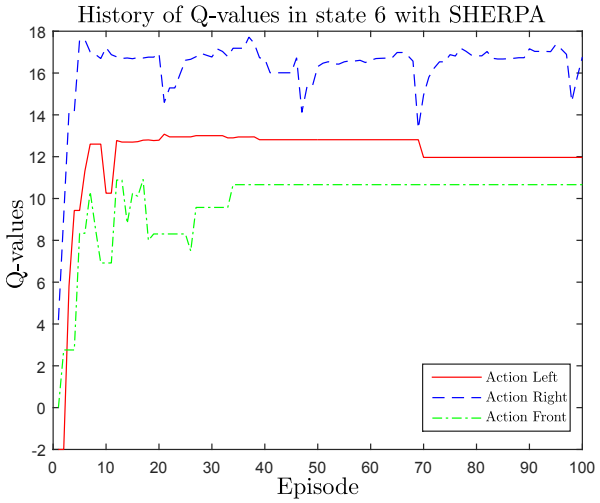
V. Comparison of Results

This section compares results of the Q-learning algorithm with and without SHERPA during the two runs of 100 episodes previously shown.

Figure 9a illustrates the comparison between Q-learning with and without SHERPA on four aspects: simulation time, number of times the goal is reached, number of collisions, and cumulative reward. Q-learning is computationally less intensive and this is illustrated in figure 9a where the Q-learning algorithm takes a fraction of the time that is required than when SHERPA is used. This is not surprising, as SHERPA must execute additional checks and iteration to determine the safety of actions and to generate backups. Comparing the number of successful runs, the agent reaches the goal more often with SHERPA than without. Therefore, for this application, SHERPA increments the efficiency of the controller. This is due to the fact that SHERPA corrects a large number of undesirable actions. This can be seen from figure 9b, which presents the total number of greedy and

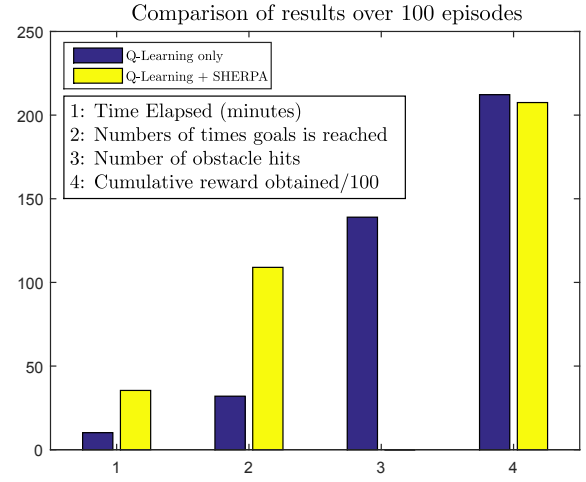


(a) Negative reward given by SHERPA

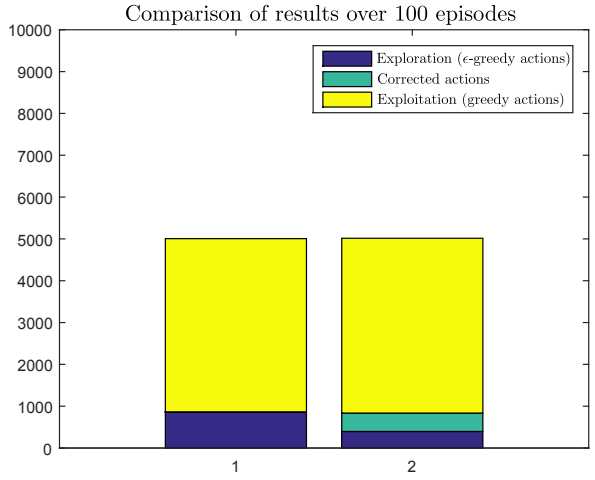


(b) Q values for state 6 $s = (2, 3, 1)$

Fig. 8: SHERPA assigns negative rewards during the entire simulation, including during the exploitation phase. This causes considerable, but temporary reduction in Q-values.



(a) The four aspects of comparison: computation time, number of times the goal is achieved, number of collisions, and total reward.



(b) Comparison of greedy, random and corrected actions.

Fig. 9: Comparison of results for the Q-learning algorithm with and without SHERPA

random actions, and the number of corrected actions when including SHERPA: roughly one half of the random actions are corrected. As a result, not only SHERPA prevents collisions, as the figure shows, but prevents the agent from wasting timesteps by attempting unsafe actions. Finally, the cumulative reward obtained during 100 episodes is slightly lower when using SHERPA. This is due to the corrective, negative reward assigned. Figure 10 shows a comparison of the reward obtained per episode for the two algorithms. It can be noticed how the reduction in reward caused by SHERPA is concentrated in the initial episodes and is considerably less significant during exploitation. Furthermore, regardless of the reduction in reward, the performance of the algorithm is not reduced but increased, as shown in figure 9a.

VI. Conclusions

This paper presents an application of Reinforcement Learning (RL) with safe exploration to an autonomous agent. The agent represents a quadrotor with the task of reaching an unknown goal position inside the environment. While doing so, the agent must avoid collisions with the walls surrounding the environment.

In order to do so, two algorithms are designed and evaluated. The first one is based on Q-learning. Direct reward is used to teach safe action to the controller, in combination with a very simple state representation, and a small action set, to accelerate learning. Although the state-action values are shown to fluctuate depending on the learning rate adopted, the policy is shown to quickly converge to the safest one. Nonetheless, collisions are reported during the initial episodes of learning,

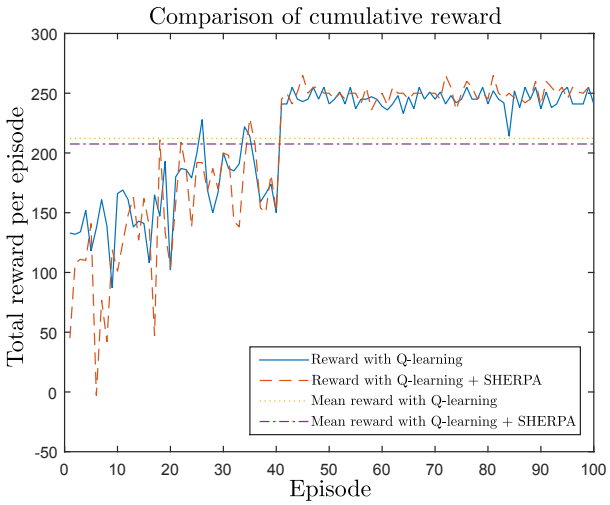


Fig. 10: Comparison of rewards obtained with and without SHERPA

due to the random actions performed during exploration.

The second algorithm evaluated is an augmented version of the first, to which the Safety Handling Exploration with Risk Perception Algorithm (SHERPA) is added. SHERPA uses interval analysis to perform near-time predictions and evaluations of actions. Only actions that are considered safe are accepted. In the event that no actions are accepted, backups are generated, using an augmented action set.

Comparing the two algorithm, the inclusion of SHERPA increases the computational effort for the controller by several times; however, the augmented algorithm is shown to completely prevent collisions, which in turn has the benefit of reducing ineffective actions, thus improving the number of successful iterations as well.

Several improvements on the paper are possible. A larger action set could be implemented to increase the learning space of the agent. An altitude control for the agent could be included to evaluate safe exploration in a three dimensional plane. Also, the proposed approach could be applied in a real world experiment, in order to investigate the effect of the reality gap. Nonetheless, the approach presented in this paper constitutes a promising result for safe exploration in model-free RL, with a focus on limited computational requirements.

REFERENCES

[1] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An introduction*, vol.1, no.1, Cambridge: MIT Press, 1998.

[2] T. Fraichard and H. Asama, "Inevitable collision states: A step towards safer robots?", *Advanced Robotics*, vol.18, num.10, pp.1001-1024 October 27-31, 2004. ing from delayed

[3] T. M. Moldovan and P. Abbeel, "Safe Exploration in Markov Decision Processes", in *Proc. 29th Int. Conf. Mach. Learning*, Edinburgh, Scotland, Jun. 2012.

[4] T. Mannucci et al., "SHERPA: A safe exploration algorithm for Reinforcement Learning controllers", in *AIAA Guidance, Navigation, Control (GNC) Conf.*, Kissimmee, FL, Jan. 2015, pp. 1757-1771.

[5] F. J. G. Polo and F.F. Rebollo, "Safe reinforcement learning in high-risk tasks through policy improvement", in *IEEE Symp. Adaptive Dynamic Programming and Reinforcement Learning (ADPRL)*, Paris, France, Apr. 2011, pp.76-83.

[6] B. Zuo et al., "A reinforcement learning based agent navigation system", in *IEEE Int. Conf. Syst. Man Cybern.*, San Diego, CA, Oct. 2014.

[7] Tamilselvi, D., Shalinie, S.M. and Nirmala, G. "Q-learning for mobile agent navigation in Indoor environment", IEEE International Conference on Recent Trends in Information Technology, June 3-5, MIT, Anna University, Chennai, India, 2011.

[8] C. J. C. H. Watkins and P.Dayan. "Q-learning", *Machine learning*, vol.8, num.3-4, pp.279-292, 1992.

[9] C. J. C. H. Watkins, "Learning from delayed rewards", Ph.D. dissertation, Univ. Cambridge, Cambridge, England, 1989.