The Actor-Judge Method

Safe state exploration for Hierarchical Reinforcement Learning Controllers

Verbist, Stephen; Mannucci, Tommaso; van Kampen, Erik-Jan

DOI
10.2514/6.2018-1634

Publication date
2018

Document Version
Accepted author manuscript

Published in
Proceedings of the 2018 AIAA Information Systems-AIAA Infotech @ Aerospace

The Actor-Judge Method: safe state exploration for Hierarchical Reinforcement Learning Controllers

S. Verbist*, T. Mannucci† and E. van Kampen‡
Delft University of Technology, Delft, Zuid Holland, the Netherlands

Reinforcement Learning is a much researched topic for autonomous machine behavior and is often applied to navigation problems. In order to deal with growing environments and larger state/action spaces, Hierarchical Reinforcement Learning has been introduced. Unfortunately learning from experience, which is central to Reinforcement Learning, makes guaranteeing safety a complex problem. This paper demonstrates an approach, named the actor-judge approach, to make the exploration safer while imposing as few as possible restrictions on the agent. The approach combines ideas from the fields of Hierarchical Reinforcement Learning and Safe Reinforcement Learning to develop a Safe Hierarchical Reinforcement Learning algorithm. The algorithm is tested in a simulated environment where the agent represents an Unmanned Aerial Vehicle able to move laterally in four directions using quadridirectional range sensors to establish a relative position. Although this approach does not guarantee the agent to never explore unsafe areas of the state domain, results show the actor-judge method increases agent safety and can be used on multiple levels an HRL agent hierarchy.

Nomenclature

HAM Hierarchical Abstract Machines.
HRL Hierarchical Reinforcement Learning.

MDP Markov Decision Process.

RL Reinforcement Learning.

SARSA State-Action-Reward-State-Action.
SHERPA Safety Handling Exploration with Risk Perception Algorithm.
SHRL Safe Hierarchical Reinforcement Learning.
SMDP Semi-Markov Decision Process.
SRL Safe Reinforcement Learning.

UAV Unmanned Aerial Vehicle.

I. Introduction

Reinforcement Learning (RL) is an exciting field of machine learning offering many applications in the fields of robotics,[1] wind energy conversion[2] and Unmanned Aerial Vehicle (UAV) control[3,4] among others. With used in physical systems that can be damaged from incorrect controlling, it becomes important to focus research on making safer while maintaining its benefits.

Inspired by animal learning behavior[5] involves an agent performing (sequences of) actions based on its observed state and obtaining a positive or negative reward from the environment. It then adjusts its strategy

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

accordingly in order to find the strategy that accumulates the highest total reward. In the absence of any additional information or instructions, flat RL allows an agent to explore every state in its domain to find the optimal strategy. This includes potential unsafe states that the agent may find in its domain, however in order to autonomously explore its state domain the agent is free to find these unsafe states. Of course one option may be to simply remove these undesired states from the state domain of the agent prohibiting it to explore them, however this assumes full knowledge of the designer on every potential unsafe state and this may not always be clear.

In order to develop further understanding into balancing safety, exploration and agent autonomy, the goal of this research is to develop an algorithm that allows an agent to explore an unknown environment with as little prior knowledge about the environment as possible. During this research, an approach is formulated and tested against other methods of state exploration with and without additional safety considerations in the RL framework.

The contribution of this paper is to present the actor-judge framework, building on the knowledge in RL to create safer and better functioning autonomous robots in a world of ever increasing interest in these systems for control.

The organization of this article is as follows. This introduction is followed by a short summary in Section II of the background fundamentals in Safe Reinforcement Learning (SRL) and Hierarchical Reinforcement Learning (HRL). Section III introduces the safe learning approach and explains the experimental setup used for testing the algorithm and comparing it to other approaches. Section II shows some more detail on the environment and agent design as well as presenting the simulation variables that will be used to compare the different approaches. The results of the simulations are shown and discussed in Section V and the article is concluded in Section VI, summarizing the major findings and drawing conclusions.

II. Fundamentals

This section presents the fundamentals of RL. Since much of the working principle of RL relies on the problem being a Markov Decision Process (MDP), this section will begin by explaining this followed by the fundamentals of RL. This section concludes with some background in HRL and SRL.

A. Markov Decision Processes

A MDP consists of the quintet $\langle \mathcal{S}, \mathcal{A}, T, \mathcal{R}, \gamma \rangle$, the states, actions, transition probabilities, rewards and discount factor respectively. When there is a finite amount of states and actions, it is said to be a *finite* MDP. When an agent finds itself in state $s \in \mathcal{S}$ at time t and performs action $a \in \mathcal{A}$, which transitions the agent into state $s' \in \mathcal{S}$ with probability $T^a_{ss'} = \mathbf{Pr}(s_{t+1} = s' \mid s_t = s, a_t = a)$, it receives a reward $R^a_{ss'} = E\left\{r_{t+1|s_t=s,a_t=a}\right\}$. The total discounted return is then $R_t = \sum_{k=1}^{\infty} \gamma r_{t+k}$ with one-step reward $r = R^a_{ss'} \in \mathcal{R}$. An important rule in an MDP is the *Markov property*, which says that each state is only the result of its immediate predecessor state and the action chosen regardless of the time-history of states and actions.[5] This property is repeated in Equation 1.

$$\mathbf{Pr}\left(s_{t+1} \mid s_t, a_t, s_{t-1}, a_{t-1}, \ldots, s_0, a_0\right) = \mathbf{Pr}\left(s_{t+1} \mid s_t, a_t\right) \tag{1}$$

This property is assumed to be valid throughout most of the research carried out. Where this property might be in violation is pointed out at the relevant section.

B. Reinforcement Learning

Reinforcement Learning is a branch of machine learning inspired by animal learning behavior.[5] It relies on experience of past rewards given to an agent by the environment in response to its actions. The goal of the agent is usually to find the action, or sequence of actions that yield the highest total return over the course of an episode. The return is defined in Equation 2 as the sum of all discounted rewards given to the agent by the environment.

$$R_t = \sum_{k=1}^{\infty} \gamma r_{t+k} \tag{2}$$

An RL agent interacts with its environment as displayed in Figure 1. The state s is a particular state of the set of all possible states $\mathcal{S}$. Similarly, the action a of all possible actions $\mathcal{A}$ is a single particular action. The agent is tasked with finding the action $a \mid s$ that yields the highest return.

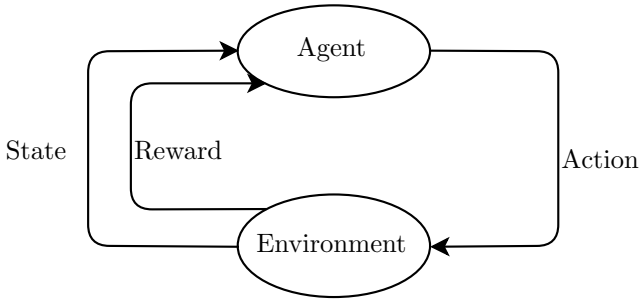


Figure 1: Agent interaction with its environment

After choosing an action, the agent receives a reward from the environment r, which is used to update the *state value function* $\mathcal{V}$ or *state-action value function* $\mathcal{Q}$. This update can be done in a variety of ways such as Q-learning,[6] Sarsa,[5] dynamic programming,[7] Monte Carlo.[8] An agent's *policy*, $\pi(s) = \mathbf{Pr}(a \mid s_t = s)$ chooses its actions based on these *value functions*. One key distinction between the various approaches is the availability of a model of the environment and the time until value update can be performed. Dynamic Programming requires a full model of the environment to work and Monte Carlo requires an entire episode be run until completion while Temporal Difference methods like Sarsa and Q-learning do not.[5] The approach in this research is such that the agent does not have a model of the environment and performs value updates after every step and it was therefore chosen to use the State-Action-Reward-State-Action (SARSA) method.

SARSA means that this update only requires the states s_t and s_{t+1}, the actions a_t and a_{t+1} and the reward r. The value update is done as shown in Equation 3 and will be central to the updates performed in this research.

$$\mathcal{Q}(s,a)_{k+1} = \mathcal{Q}(s,a)_k + \alpha(r + \gamma * \mathcal{Q}(s_{t+1}, a_{t+1})_k - \mathcal{Q}(s,a)_k) \tag{3}$$

B.1. Hierarchical Reinforcement Learning

One area of research in this paper is the benefit to safety added by using HRL. HRL is often used as a way to deal with the *curse of dimensionality*[9] in problems where the state- and/or action spaces $\mathcal{S}$ and $\mathcal{A}$ respectively, become too large to search through efficiently. HRL constructs a hierarchy of sub-spaces that become MDP problems by themselves under the assumption that solving each sub-problem will also solve the whole. An added benefit to this is that an agent can be given *strategies* for solving a problem. This is used in the approach developed in this research and will be explained more thoroughly in Section III.

In principle, HRL allows for an agent navigating in a house, to learn sub-problems such as *exit a room* or *open door*. The root (top-level) RL problem then becomes to use these actions of exiting a room and opening a door to navigate through the rooms without having to learn how to exit each individual room separately.

The main approaches covered in literature are Parr and Russel's *Hierarchy of Abstract Machines* (HAM),[10] Sutton et. al's *Options* framework[11] and Dietterich's *MAXQ*.[12] The ideology behind each approach is quite similar such that they each split the main task up into smaller pieces to be solved separately under the assumption that solving them individually, will solve the original task.[9]

B.2. Safe Reinforcement Learning

SRL has only recently been coined by García and Fernández in their survey on safe reinforcement learning.[13] Their definition of SRL is interpreted as any approach that considers safety and/or risk in RL algorithms. While pursuing safety might not have been the goal in some approaches covered in the survey, it does raise an important concern about RL agents and safety. RL is based on learning by experience and is therefore implicitly unsafe. A unknowing and inexperienced agent will not be able to tell a safe state apart from an unsafe one until it has experienced both of them at least once. The approaches covered in the survey all share in common that they augment the agent in a way that gives it more information, either by adjusting the possible actions the agent may take, or provide a-priori information of the environment. Another approach is to simply perform a (simulated) trial run before committing to a physical system.

While initially a problem solver for the curse of dimensionality, HRL is also being explored as a viable approach to SRL.[14] The benefit is clear when considering the house-navigating agent from the previous section. An agent learning how to navigate one room will also learn that colliding with walls is sub-optimal. The agent can reuse this information avoiding collisions for any other similar room without having been there before which can be interpreted as SRL. Safety can therefore be noticed as an additional benefit of HRL even when it was not the goal.[15]

Mannucci et al. developed a type of action filter called Safety Handling Exploration with Risk Perception Algorithm (SHERPA) for RL controllers that discovers fatal states while running and provides backup actions that can steer the system back to safety.[16] This algorithm is designed such that it can be used by an RL controller. The approach covered in this research paper draws inspiration from SHERPA and the various approaches covered in the survey to develop an algorithm that limits the prior knowledge given to the system while also not altering the state or action space of the agent.

III. Agent algorithm design

As explained in Section I, the contribution of this paper is to formulate an approach adding safety to exploration in the RL framework coined the *actor-judge method*. This section begins with a general explanation of the Judge followed by a description of the developed algorithm. In addition, this section describes the agent Hierarchical Abstract Machines (HAM) structure and finally, a general description of a task designed for testing the capabilities of the developed algorithm.

A. Actor-judge method

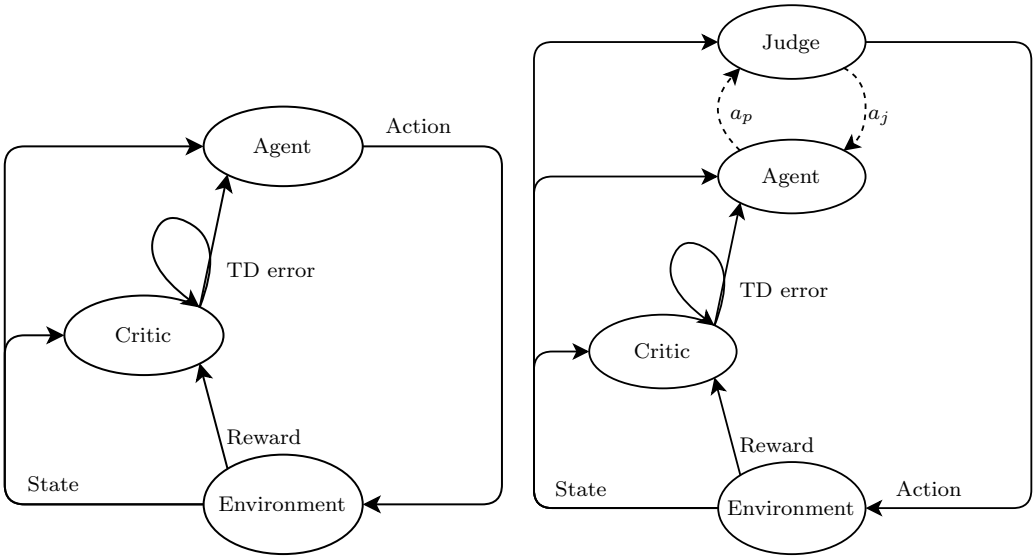
Taking inspiration from the actor-critic model,[5] teacher advice[13] and SHERPA,[17] the 'actor-judge method' is presented. The actor-critic method has shown to improve learning rates by reducing the number of irrelevant states visited.[18] The agent interacts with the judge somewhat similarly to teacher advice where the judge interacts with the agents' proposed actions. Likewise SHERPA proposes a type of 'filter' that checks the proposed action for potential unsafe behavior as a result.[17]

A judge has been developed in this research as a combination of these approaches approaches. Like the teacher, it can communicate with the agent and provide limited extra knowledge (or rather, more considerations) and like SHERPA, it behaves more like a 'safety filter'. The judge is a modular addition to the RL agent as can be seen in Figure 2 where the judge has been attached to an actor-critic setup. It receives the proposed action a_p from the agent and if the judge overrules this, returns the judged action a_j. It also passes the selected action to the environment. In the actor-critic method, Figure 2a, the critic receives the reward and updates its value function while the agent chooses actions based on its policy. Note that in the judge-action method, Figure 2b, the judge does not receive the reward from the environment but uses the state information to update its own behavior.

A benefit to this simple setup is that the judge can be created as an entirely modular component of the agent-environment interaction. This is useful on a multilevel machine learning setup such as HRL where safety might be a concern on various levels. In this research, safety is primarily a concern on the primitive level i.e. on the level where actions interact with the environment, but this does not always have to be the case as is demonstrated briefly in Section V where the judge has been applied on the root level concerning itself about the agent's battery level. This modularity is shown in Figure 3.

In the setup presented, the judge is initialized with backup action $a_j \in \mathcal{A}$ where a_j is the action that the judge uses to respond to potentially dangerous situations. Note that the knowledge of these actions violates some autonomy of the algorithm as a whole and a safe action a_j might not always be known. In this research, the existence of this safe action is assumed and considered somewhat analogous to instinctual *fight-or-flight* behavior in animals. In order to maintain autonomy, one might be able to find this action using other forms of machine learning (perhaps something akin to the way animals have evolved their response to dangerous situations).

The working of the judge is shown in Algorithm 1. The judge will belay the agent actions with a probability $\mathbf{Pr}_j(a_t = a_j \mid s_t = s)$ which is initialized with j_0 for every new state. This probability changes based on the occurrence (or not) of unsafe events or states. The probability updates are shown in Equation 4 where ϕ and ρ are the *forgiveness* and *punishment* factors respectively. The forgiveness factor $0 < \phi < 1$ is multiplied with the overruling probability if nothing undesirable occurred or the punishment factor $\rho > 1$



(a) Actor-critic method (b) Actor-judge method

Figure 2: Diagrams of the actor-critic and actor-judge method.

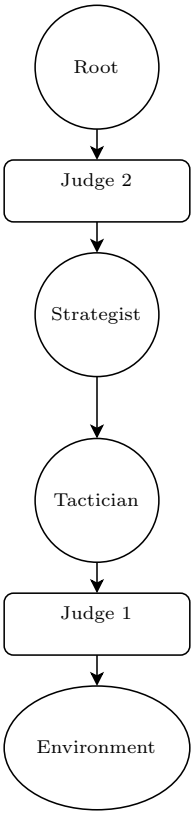


Figure 3: Diagram showing how the actor-judge method can be interjected on various levels in a HRL learner.

otherwise.

It may occur that an agent finds itself m number of times in a particular state s where $m \gg 1$ which would cause the overruling probability for this state to become $\mathbf{Pr}_j(a_t = a_j \mid s_t = s) \to \lim_{m\to\infty} \phi^m \cdot j_0 = 0$. The problem here is that when the agent eventually does reach another state, it could be an unsafe one and the agent would then again update the overruling probability as $\mathbf{Pr}_j(a_t = a_j \mid s_t = s) = \rho\phi^m j_0 \approx 0$. This means it will take n amount of unsafe transitions where $n \gg 1$ until $\mathbf{Pr}_j(a_t = a_j \mid s_t = s) = \rho^n\phi^m j_0 \approx j_0$ which is unacceptable in systems where safety is crucial. In order to prevent this, the overruling probability is given a lower bound $\underline{j}$. This number can be configured to whatever is needed or be made adaptable based on the task. For the simulations in Section III, this value is fixed $\underline{j} = 1e-3$. It should also be noted that the judge assumes that, if an unsafe state or event has occurred, it is due to the agent not having performed the backup action. The agent also receives back the judged action and can use this to update its value function internally. The agent uses a reward of $0.5 \cdot r$ to update its value function with the proposed action if this is not equal to the judged action.

```
1  Initialize backup action and unsafe state, action or event ;
2  while Performing the task do
3  |  Receive state and agent proposed action ;
4  |  Update judge overruling probability ;
5  |  if State unknown then
6  |  |  Initialize overruling probability Pr_j(a_t = a_j | s_t = s) = j_0
7  |  end
8  |  if Proposed action is not backup action then
9  |  |  Replace agent action with probability Pr_j(a_t = a_j | s_t = s) ;
10 |  end
11 end
```

Algorithm 1: General algorithm of the judge

$$\mathbf{Pr}_j(a_t = a_j \mid s_t = s) \leftarrow \begin{cases} \phi \cdot \mathbf{Pr}_j(a_t = a_j \mid s_t = s) & \text{if (safe transition)} \\ \rho \cdot \mathbf{Pr}_j(a_t = a_j \mid s_t = s) & \text{if (unsafe transition)} \end{cases} \tag{4}$$

The actor-judge algorithm is illustrated with an example. Figure 4 shows an agent (circle) in a 3-cell environment with states s_1, s_2, s_3 from left to right as the cells that the agent can occupy. The filled square represents a wall whose cell the agent cannot enter. In episode 1, the agent is initialized in the middle cell s_2 at t_0. Upon selecting an action, this gets proposed to the judge. The judge receives this action in line 3 of Algorithm 1 followed by an attempt to update the overruling probability of the previous state in line 4. However, because this is the initial state, this part has no effect. The state s_2 is unknown to the judge who assigns it an overruling probability of $\mathbf{Pr}_j(a = a_j \mid s = s_2) = j_0$ in line 6, and in line 9 the agent's action is overruled using this probability.

At time t_1, the agent is progressed to the next cell s_3 and proposes an action to the judge. After the judge receives this action, it now updates the probability $\mathbf{Pr}_j(a = a_j \mid s = s_2) \leftarrow \phi \cdot j_0$ if this state has been reached safely. s_3 is also unknown to the judge so this is initialized $\mathbf{Pr}_j(a = a_j \mid s = s_3) = j_0$ and subsequently overruled with probability j_0. Assuming the agent managed to try and move to the right again, this would result in an unsafe action.

Thus at $t = t_2$ the agent's next action is received and the agent's choice at t_1 is punished as $\mathbf{Pr}_j(a = a_j \mid s = s_3) \leftarrow \rho \cdot j_0$. Since the agent has been in s_3 before, line 6 of Algorithm 1 is skipped.

An important assumption for using the actor-judge algorithm is that an agent is allowed to approach unsafe states slowly. The behavior that will be exhibited by the agent will be that unknown states will be treated as unsafe and the judge will overrule the agent's proposed action. However over time, as the agent visits the same state and proposes the same action with no unsafe outcome, the agent will be allowed to perform its original action, potentially resulting in an unsafe state. If the reward function has been created such that this state results in a large negative reward, the agent will be dissuaded to perform this action again (as is usual in RL).

Although the agent has still reached a potentially harmful state, the judge can either postpone this event to the point where the agent has learned optimal behavior safely, or allow the agent to *slowly* approach

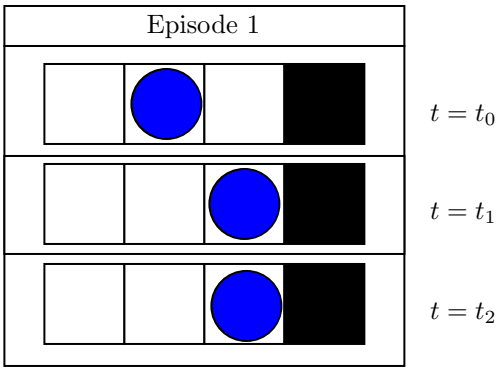


Figure 4: Three time-steps an episode to illustrate the working principle of the actor-judge method.

unsafe regions of its state space. For example a UAV may be allowed to bump into a wall with a low velocity, which is unwanted behavior, but not immediately destructive like crashing full speed instead.

Should there be unsafe actions that will instantly cause irreversible harm to the agent, the actor-judge method as shown above cannot prevent it, neither is it expected to. In this manner, the agent explores its environment while maintaining a level of autonomy that would not be had if the agent was simply forbidden from accelerating towards a wall.

B. Task description

As a means to test HRL as a means of SRL with the added benefit of a judge, a task is designed for the agent to perform. The task of the agent is quite similar to a lot of RL problems: find the quickest path to the goal. The environment in which the agent has to perform its path-finding is one governed by the laws of motion which adds some complexity to the task for the value function updates.

In addition, because safety is the primary concern of this research, the agent is given some way of sensing its environment much like a real UAV. The task therefore does not provide the agent with its absolute position within the environment but instead its position relative to its surroundings with range sensors further adding complexity to the task. Of course, in order to determine safety, it needs to be defined for the task what exactly is 'safe'. In this navigation task, unsafe behavior is defined as colliding with walls; the higher the speed, the more unsafe it is. While colliding with the wall at any speed is considered an unsafe event, soft collisions will be considered acceptable while the agent explores its environment.

Although traditional RL should eventually learn 'safe' behavior with a proper reward function, the goal here is to allow collisions under some speed $\overline{v}_{safe}$ into the wall that the agent may use to learn that obstacle collision is unsafe. A distinction made here is that a traditional RL controller, upon colliding with a wall at some speed, may conclude that the reason for a negative reward was that it wasn't approaching it fast enough. This is of course an undesirable conclusion on an expensive physical system that can break at high velocity impacts. Note that although in order to test the safety benefits of the actor-judge method it is not needed that there is a goal at all, a goal state is provided to test the impact on the learning speed as well.

C. Agent definition

The actor-judge method was developed as a means of SRL for a HRL controller but the description has been left as a general approach up to this point. In order to encourage safe behavior in an unknown environment with the agent representing a UAV with range sensors, the agent has been constructed as a HRL controller. Preliminary results have also shown the much improved behavior of HRL agents in a task with relative state awareness. This also allows for structuring the hierarchy to incorporate strategies that the agent can utilize to navigate its environment. Additionally, a flat RL agent with this limited positional awareness would violate the Markov property as the next state cannot always be known without the time history of states and actions. HRL with navigational strategies given to the agent would allow for the assumption of Semi-Markov Decision Process (SMDP) where the states for the strategies are temporally abstracted.

The agent's hierarchy is structured according to Parr and Russel's HAM[10] and has been designed as shown in Figure 5. The root note at the top can choose strategists to realize high level goals. Each strategist in turn

can select different strategies or tacticians that are also separate machines or temporally extended actions. Finally the tactician selects the primitive actions (tactics) available to it to find the optimal (sequence of) actions to complete the sub-goal of the strategist it is enacted by. It is assumed that if the tactician finds the optimal primitive actions to complete the strategy subgoal and the strategist finds the optimal (sequence of) strategies to complete the sub-goal of the root node, then the root node has optimally completed the overall task.

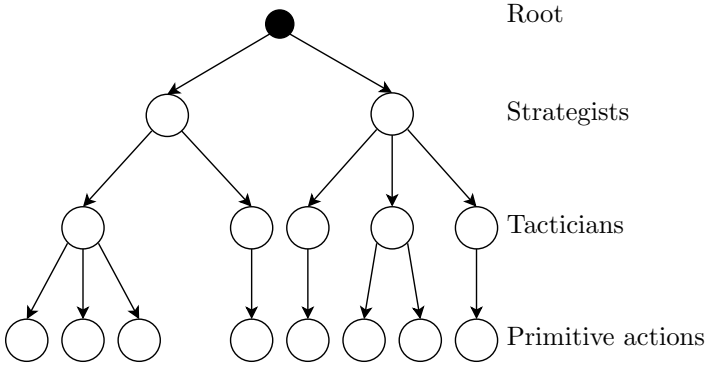


Figure 5: Structure of the HRL controller

The algorithm used to train this HRL agent is shown in Algorithm 2.

```
Initialize states, actions and Q-values ;
repeat
    Perform a_root repeat
        Perform strategy ;
        repeat
            Perform tactic, observe reward and next state ;
            Select next tactic ;
            Update tactic Q-values ;
        until s_tact is terminal;
        Observe reward and state ;
        Select next strategy ;
        Update strategy Q-values ;
    until s_strat is terminal;
    Observe reward and state;
    select new root action ;
    Update root Q-values ;
until s_root is terminal;
```

Algorithm 2: HRL pseudocode algorithm with using Q-value function updates to train an agent

IV. Simulation design and application

Before presenting simulation results in Section V, this section will provide a little more detail on the simulation framework and variables. This section begins with an explanation of the simulation framework that was developed wherin the environment for the task is created. Next is a description of the environment variables that are changed between runs to observe the effect of these variables.

A. Simulation framework

This research has been conducted entirely in a simulated environment, developed and written in the programming language Python. The simulation will consist out of an agent, representing a UAV flying in a two-dimensional environment governed by the laws of motion. The agent controls force vectors in four directions in the horizontal plane in order to move itself.

The environmental updates, i.e. changing the state vector of the agent due to dynamics, happen independently of the agent input. The agent perceives the environment and calculates an action every command interval of T_i seconds. The environment continuously updates the agent position based on the last known agent action using simple equations of motion. The agent actions are translated to force updates which in turn are twice integrated to position using dead reckoning.

The handling of the communication between the environment and agent works as follows; the environment always generates a state for the agent and adds this to a so called *Queue*, which acts like a mailman that can send messages between the agent and environment. This mailman returns every next update and if the state has been accepted, the queue is empty; if not, the state is discarded. Similarly there is a queue for the actions that the agent sends to the environment which the environment considers the agent's command until a new action is sent. It is only at a set interval that the agent sends an action to the environment, which the dynamics equations will use to alter the state of the agent. This has been chosen to more closely relate the simulation to a real world example where; instead of the environment waiting for the agent to deliver an action and processing it, the environment is continuously updating with or without agent input. The flow of the simulation is displayed in Figure 6. In this figure, the agent and environment each have a program 'thread', represented by the dotted line, which shows the processes run on each thread, represented by small red rectangles. What can be seen here is that the environment updates much more frequent than the agent and as such, the agent only uses the state of the environment at the start of the agent process to determine an action. This means that any other state update after the last agent action update and before the last state pickup, is ignored.

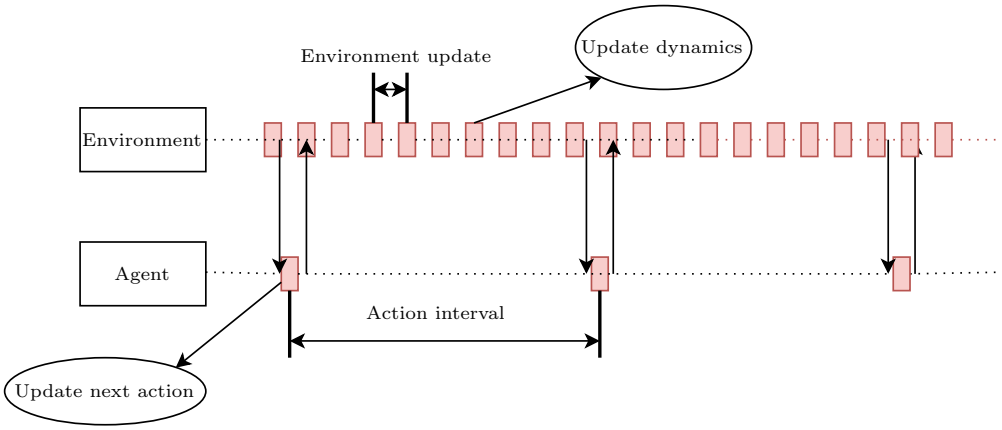


Figure 6: Flow of simulation. The environment gets updated continuously and periodically receives an action from the agent

In this environment, there is also a simple object collision and resolution mechanic that simply causes the agent to 'bounce' off of objects at a reduced speed. In order to determine the effects of safety, it is of course necessary to have a clear definition of safety. In this environment there will be two aspects that are considered 'unsafe'; bouncing into a wall and running out of battery. The goal is to develop an algorithm that will steer the decision process of an untrained agent faster towards actions that will avoid ending up in a bounce or running out of battery while minimally exposing itself to these unsafe states.

An agent performing an action that results in a bounce or finding itself with a depleted battery will yield a reward of -10. Reaching the goal will yield a reward of 0, while all other actions get a -1 reward. The environment is shown in Figure 7 with the agent represented by the circle and the goal in the lower-right as the rectangle with the triangle inside it. The total size of the environment is arbitrarily chosen at 25x25 meters (a decent-sized building interior) and the agent has a maximum speed of 3 m/s.

B. States and Action Space

This section provides more detail of the state and action space that the agent will be allowed to navigate. It is assumed that the agent is a simulated model of a UAV that only has one range sensor in four directions as shown in Figure 8. This is due to the commitment of creating a simulation that more closely resembles a real-world example where absolute position is often unknown in an unknown environment.

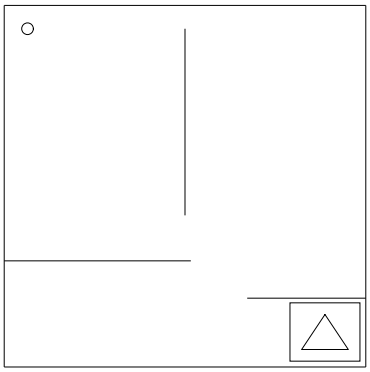


Figure 7: The environment where the agent (circle) has to navigate through. The rectangle with the triangle in the lower right is the goal.

In order to provide faster learning and avoiding the curse of dimensionality, these distances measured by the range sensors are discretized into a number of discrete ranges or sensor resolution. There will be tests with a sensor resolution of 3 where the numerical distance from the range sensor of agent is mapped to discrete values "Far", "Close" or "Critical".In addition to sensor ranges, the state also consists of the agent's velocity in the global x- and y-direction. For when it is relevant, the agent will also track its battery level and distance and direction to recharge point. Note that there will be simulated runs when the battery level is not taken into account to compare the impact on performance when battery is to be considered. When the battery life is of interest, the environment will lower the battery % over time, but maintain it at the lower bound of 10%, providing a negative reward to the agent until the agent has recharged its battery.

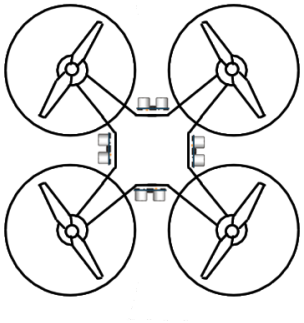


Figure 8: Drone sketch showing the location of the four range sensors

The flat RL and random-action agent will have 4 primitive actions available to them at any given state which will accelerate them in the chosen direction:

- Move up
- Move right
- Move down
- Move left

In the hierarchical setup, the root node at the top, Figure 9a, decides on exploring and learning about its environment, or exploiting a fixed fuel-finding policy by selecting strategists to realize these sub-tasks. The goal of the exploration strategist is to find the shortest path to the goal and it can select temporally extended actions, or strategies that are separate RL machines. These strategies are similarly named as the flat RL primitive actions shown in Figure 9b. However in this case, these strategies are temporally extended actions and will terminate when the agent is stationary near a wall (distance to wall "Critical"). Each strategy is

in turn enacted by a tactician which is again a separate RL machine, however the tactician chooses between primitive actions. For example, the explore strategist wants to enact the strategy 'Up'. The tactician will use the primitive actions (tactics) shown in Figure 9c to find the optimal sequence of actions to the satisfaction of the strategist machine.

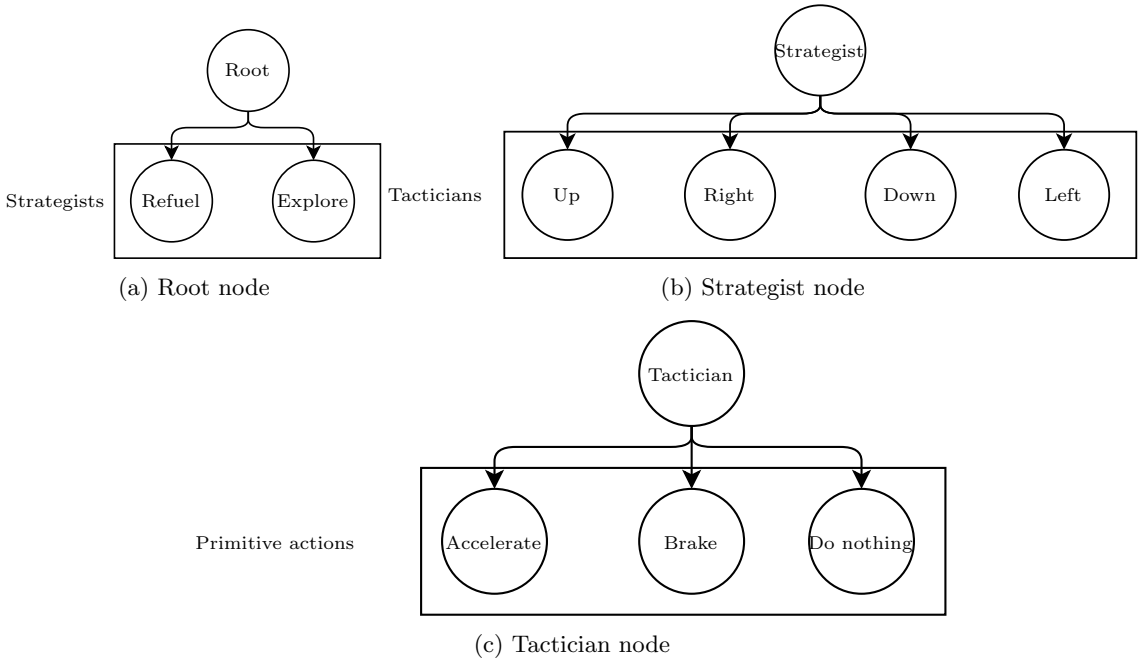


Figure 9: Structure of the HRL controller

C. Simulation Variables

Here some information is given regarding the variables that are set in the simulation. For comparison between approaches, there will be four different actors attempting the task:

- Random action agent
- Flat RL agent
- HRL agent
- HRL agent augmented with the actor-judge method

A common setting is established for all but the random action agent with fixed simulation settings as shown in Table 1. These simulation settings are specific to RL and are thus not relevant to the random action agent. Because randomness is a factor in the decision process of the agent, each simulation run will be performed three times with the same settings to reduce the luck factor. This however excludes the random action agent which will only perform the task once since there is of course no learning at all. The random action agent will also only be tested once as a baseline.

To determine their impact on the results, these settings are changed between simulations. Augmented with the actor-judge method, the Safe Hierarchical Reinforcement Learning (SHRL) agent will be tested several times with various judge settings. The goal is only to determine their effect, not the optimal settings and therefore the settings are only changed a limited amount of times.

The different simulation settings will be tested and compared with each other in the following ways:

- Number of collisions per episode
- Number of total collisions
- Collisions above v_{safe}

Table 1: Baseline simulation settings

Agent policy	$\pi(s,a)$	$\epsilon - greedy$
	ϵ	0.05
Maximum velocity	v_{max}	3 m/s
Maximum 'safe' collision velocity	v_{safe}	2.5 m/s
Agent mass	m	0.42 kg
Control force	f	0.25 N
Control input interval	T_i	1 s
Initial judge probability	j_0	0.95
Judge punishment factor	ρ	1.5
Judge forgiveness factor	ϕ	0.9

- Approximate episode until convergence to optimal path
- Number of actions per episode
- Judge interventions per episode

Comparing the difference in the above listed attributes will give an overview how each different approach performs and allows for conclusions to be made regarding their influence on safety and convergence. It is expected at this point, using the random agent as a baseline, that all approaches will perform better than the random agent in terms of faster convergence and more safety. Also it is expected that flat RL will be outperformed by HRL and the developed algorithm as the latter two are given additional tools that will decrease 'same-state probability', i.e. the probability that a subsequent state is the same as its predecessor $\mathbf{Pr}(s_{t+1} = s \mid s_t = s, a_t = a)$.

V. Results

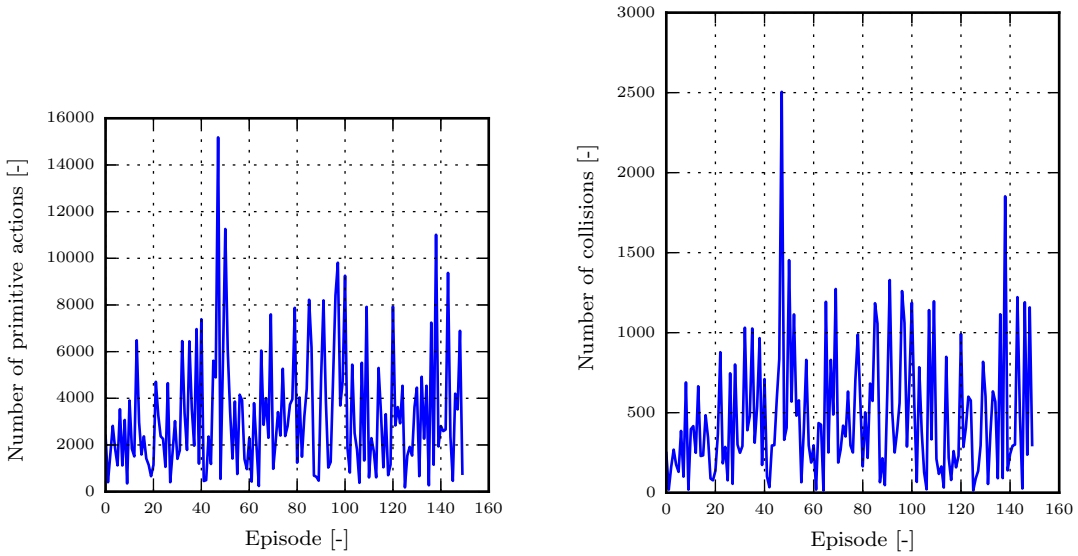
This section shows and discusses the results of the simulations performed according to the setup as described in Section IV. Beginning with an agent performing random actions, a baseline performance is established. Only the number of actions and collisions per episode is recorded here as there is no return for the agent here. Next is the flat RL agent showing a baseline performance for RL to which HRL and SHRL are compared. The section continues with the performance of the HRL agent. Following this with the SHRL agent which also presents a table summarizing the results for a clear overview of the different approach performances. After the different approaches have been presented, the actor-judge method is further investigated by changing judge settings and noting the performance changes. Finally the section concludes with a demonstration of the actor-judge method's modularity.

A. Random action agent

The task is first attempted by an agent that performs only random primitive actions. This serves only as a minimal benchmark that any algorithm must surely beat and to give an indication also how much better a given algorithm performs. The agent chooses every action interval an action from the primitive action space $\mathcal{A} = \{UP, RIGHT, DOWN, LEFT\}$. This agent collides with the obstacles a total of 71602 times in the entire trial of 150 episodes where an episode lasts from initialization until the goal state has been reached. Additionally it takes the agent an average of ≈3278 primitive actions to complete an episode. This trial is of course not expected to show improvement over time but the plots in Figure 10 provide a good starting point to compare the shape of the primitive actions plot (Figure 10a) and collisions per episode plot (Figure 10b) with subsequent RL algorithm performances.

B. Flat RL

Figure 11 shows the result of 3 independent runs of a flat RL agent performing the task. The reward, number of actions and collisions per episode has been averaged and is shown in Figures 11a, 11b and 11c respectively.



(a) Number of actions of the random action agent (b) Collisions per episode of the random action agent

Figure 10: Performance of agent choosing random actions

Additionally, the run with the least amount of collisions is shown in Figure 11d. Each dot on Figure 11d shows both the episode in which the collision occurred and the velocity at which it happened.

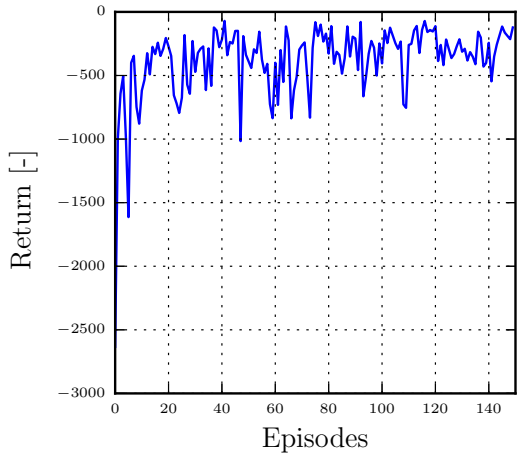
The learner does appear to show signs of converging to an optimal policy as the reward function starts deeply negative but rises towards 0. Also the 'collision density' appears to be decreasing over time. This behavior of the average return per episode is fairly consistent with the behavior of other RL tasks performed in various other literature which confirms the proper implementation of the algorithm and additionally already proves that RL can be applied on tasks where an agent only has state information relative to its surroundings without sacrificing learning.

Shown in Figure 11c, safety is not at all guaranteed for this learner which is not surprising as there was nothing in place to stop this agent from colliding with walls other than a reward function. The agent starts the first few episodes with no regard for its own safety and collides with walls almost on every approach to a wall. This behavior seems to improve over time as around episode 40 and 80, the collision density is lower although near episodes 60 and 110, the agent has reached peaks of collision density. Relapsing back into unsafe behavior such as this may be caused by a reward for collision that is too high; the agent performs a lot of actions that may not result in another state e.g. the agent may be 'far' away from a wall, perform accelerate and still be 'far' the next update. If the agent finds itself in these situations often enough, the reward given to all other alternative actions may become less desirable to the agent than attempting a collide with a wall. Lowering the reward for collision even further will likely help in deterring the agent from selecting actions that result in collision. It does seem that eventually the agent learns from its mistakes and around episode 150, the collision density appears lowest.

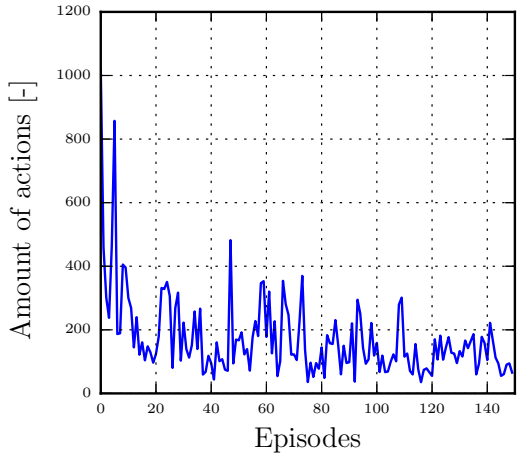
Counting the amount of collisions in Figure 11d will show that the best trial has 3956 collisions of which 724 occurred at speeds between 2.5 m/s and 3 m/s which could represent high enough speeds to cause physical damage. Amongst the three trials of this agent, there was an average of 4380 total collisions per trial. Regardless of its performance, this gives a good baseline to compare other approaches to.

C. HRL controller

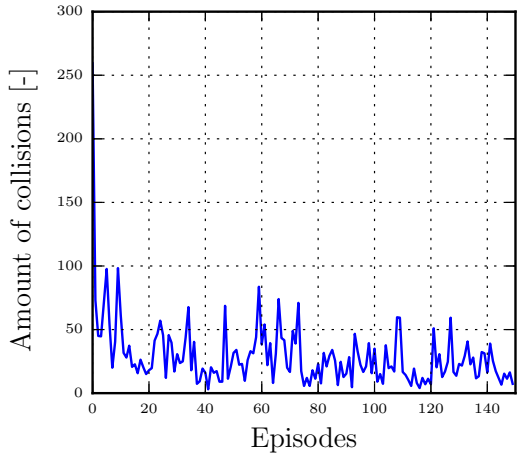
Figure 12 shows the performance of the HRL controller on the task without any additional safety considerations. The first difference compared to the flat RL controller is on Figure 12d showing a lot less density of collisions. However there are still a number of collisions at the maximum velocity which would still be undesirable in a physical system.



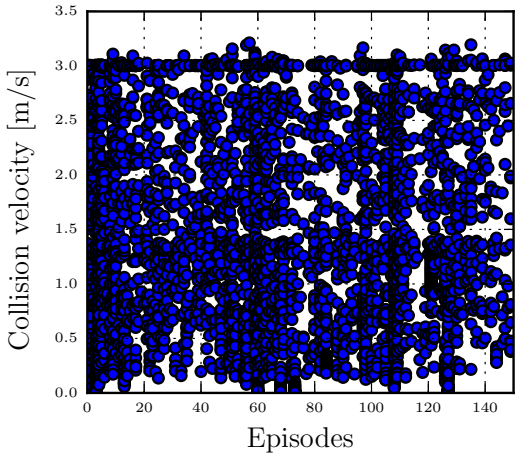
(a) Average reward



(b) Average number of actions



(c) Average number of collisions



(d) Run with the least amount of collisions, showing the velocity of each collision

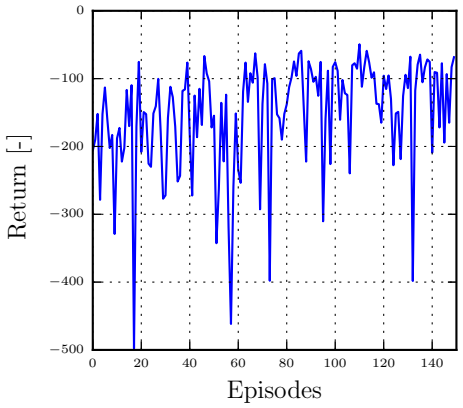
Figure 11: Performance over 3 runs of the flat RL agent performing the task

The average return here is the sum of the reward given at the primitive action level. This gives a good impression of its performance to compare it to the flat RL agent. Looking at Figure 12a, there is significant fluctuations in performance but overall appears to have an increasing trend. Comparing it to the return of the flat RL agent's case, the HRL agent seems to have learned faster, reaching the average return of -200 already within the first few episodes while the flat learner reached this point only after episode 40.

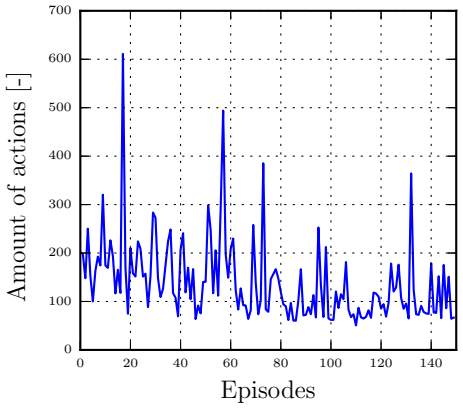
What is more of interest to this research is the comparison between the two in terms of safety. Comparing Figures 11c and 12c with each other, it is clear the HRL learner collides with its environment a significant amount of time less. Tallying the amount of collisions for the HRL learner over the three trials it shows to have reduced the average total amount of collisions from 4380 to 348; a reduction of over 92 %.

Note that these results are averaged over 3 trials each so the possibility of this simply being a 'good' run for the HRL is reduced. This seems to indicate that HRL itself is already quite a promising approach to SRL. Strengthening this argument is Figure 12d, which shows an exceptionally large decrease in collisions. Also seen on this figure is the speed at which these occur and note the decrease of collisions near maximum speed $v \geq v_{safe}$. Between episodes 80 and 150, the amount of collisions at maximum speed has decreased compared to between episodes 1 and 80.

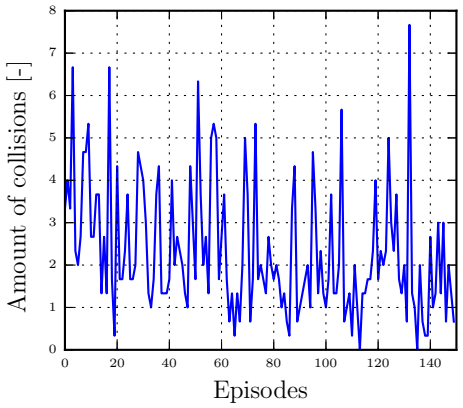
If a navigational control system should be designed for a UAV, it could already be suggested to use this learner as an option and pre-learn the agent in simulations before adding it to a physical system. This of course if few and/or soft collisions are allowed and not immediately destructive to the controlled system. But compared to a flat RL learner it is already a significant improvement.



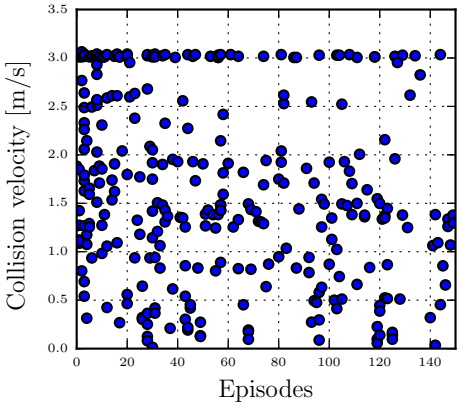
(a) Average reward



(b) Average number of actions



(c) Average number of collisions



(d) Run with the least amount of collisions, showing the velocity of each collision

Figure 12: Performance over 3 runs of the HRL agent performing the task

D. Actor-judge method

The HRL learner is now augmented with the actor-judge method on the lowest level to judge primitive actions. Figure 13 shows the overall performance of this SHRL learner. Comparing the average return over the three trials with that of the HRL learner without this augmentation shows that performance in terms of optimality is not much affected. Learning appears to occur in a similar speed as HRL and learning faster than the RL learner albeit including the same fluctuations in return as with the HRL case.

One significant difference however is shown in the collision density plot in Figure 13c where it can be seen that the density is again much lower than flat RL, and lower still than the HRL. Even more noticeable is the lower amount of collisions near maximum speed $v_{safe} \leq v \leq v_{max}$. The agent is not withheld from exploring its state space or given prior instruction on how to behave in certain conditions, maintaining some level of autonomy but was instead stopped by the judge to explore unknown states too fast. This gives the RL algorithm time to learn the more likely safe actions, while still allowing it to slowly approach a dangerous situation (a wall collision). Once a collision occurred, the agent has already learned more about its safer alternative actions to employ them instead the next time even though it could still try these unsafe actions.

Comparing the total amount of collisions with the HRL agent, the actor-judge method further decreases the average amount of collisions over the three trials to 250 from 348 which is almost a 30 % reduction.

While the addition of this system has not withheld the agent entirely from crashing full-speed into walls, it did reduce the occurrence which indicates a viable approach for safe navigation based on RL. The judge in this case is initialized with starting conditions $j_0 = 0.95$, $\phi = 0.95$ and $\rho = 1.5$ which may have more impact on the safety of the system. Of course a judge with starting conditions $j_0 = 0$ is no different from a non augmented HRL learner so some more experiments are carried out varying these starting conditions to learn their effect on the safety and/or performance.

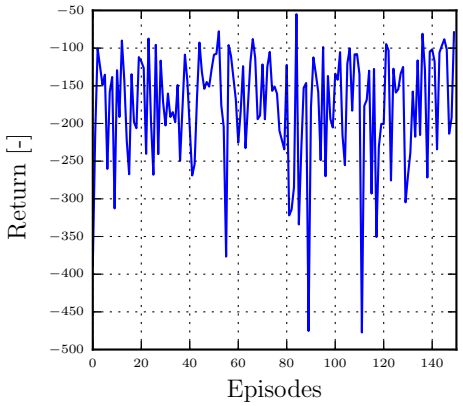
E. Summary of the results

Before looking at different settings for the judge, the results of the previous experiments is summarized in Table 2 to provide a clear overview of the performance. As described in the previous sections, the average amount of collisions is highest for the flat RL agent at an average of 4380 collisions per episode. This drops for the HRL agent to about 348 and further decreases to about 250 for the SHRL agent. What is also seen is that the average return is nearest 0 for the HRL agent however both the HRL and SHRL agents outperform the flat RL agent.

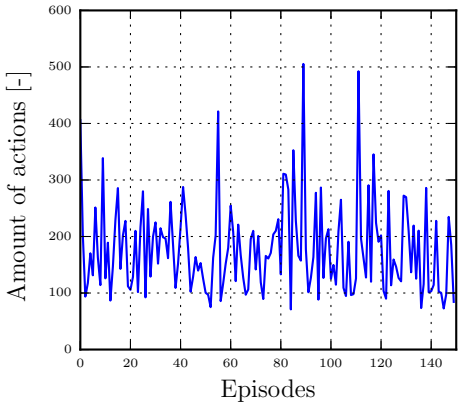
Another point of interest is the amount of collisions where $v \geq v_{safe} = 2.5$m/s where v_{safe} is considered a velocity near maximum where collisions become a danger to the integrity of the UAV. Here the absolute number of collisions above this velocity are compared which shows the flat RL agent performing worst and SHRL best. Another way of looking at this is the average ratio of the number of collisions where $v \geq v_{safe}$ over the number of total collisions. Note that this is **not** $\frac{\text{average number of collisions where } v \geq v_{safe}}{\text{average number of total collisions}}$. It is instead the ratio calculated per episode and averaged over the episodes and then averaged over the trials. Doing this calculation gives a better understanding of how many collisions happen at $v \geq v_{safe}$ per collision. For flat RL, this value is 0.22 and drops to 0.13 and 0.11 for the HRL and SHRL agents respectively. What this indicates is that these agents become less inclined to collide with obstacles at higher velocities and thus are both better performing agents in terms of safety than the RL approach. SHRL performs about 15% better on this averaged result than the HRL.

Table 2: Summary of the experiment results for the different agents.

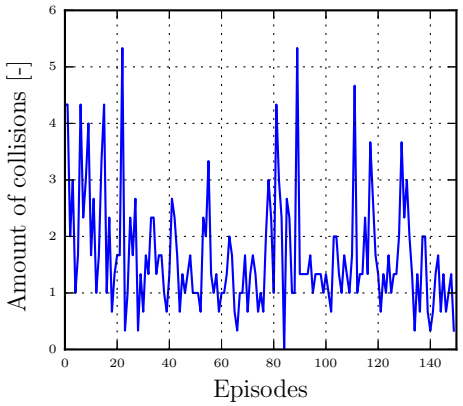
Agent	Flat RL	HRL	SHRL
Average return	-379.2	-151.5	-176.3
Average amount of collisions	4380	348.3	250.3
Average amount of collisions $v \geq v_{safe}$	845.3	70	66.7
Average ratio	0.22	0.13	0.11



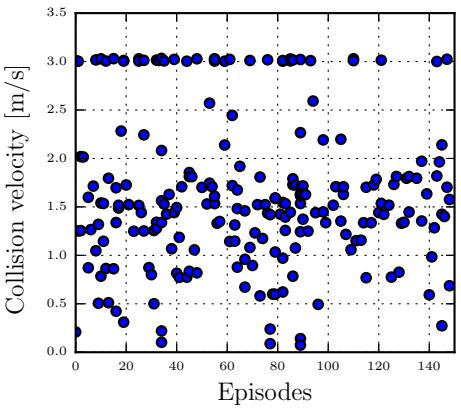
(a) Average reward



(b) Average number of actions



(c) Average number of collisions



(d) Run with the least amount of collisions, showing the velocity of each collision

Figure 13: Performance over 3 runs of the HRL agent performing the task augmented with the actor-judge method

F. Varying judge initial conditions

This section shows the results of various initial judge conditions. The judge settings as they are shown in position 1 of Table 3 are the settings used for the actor-judge agent in previous section. This section will compare other settings to this baseline to quantify the effect of the different judge settings of overruling probability j_0, the forgiveness factor ϕ and punishment factor ρ. Again three trials are run on the same settings in an attempt to lessen the randomness of performance and the results are averaged in the table.

Table 3: The averaged results over three trials of various different settings of the judge initial conditions

No.	Initial conditions	Collisions	Collided with $v > 2.5$ m/s	Return	Interjections
1	$j_0 = 0.95$, $\phi = 0.95$, $\rho = 1.5$	250.33	156	-176.27	43.67
2	$j_0 = 0.95$, $\phi = 0.45$, $\rho = 1.5$	247.33	184.67	-152.83	54
3	$j_0 = 0.95$, $\phi = 0.05$, $\rho = 1.5$	191	113.3	-127.2	25
4	$j_0 = 0.25$, $\phi = 0.95$, $\rho = 1.5$	222.67	140.3	-173.67	138.67
5	$j_0 = 0.95$, $\phi = 0.95$, $\rho = 2$	177.3	38.3	-144.9	163.7
6	$j_0 = 0.95$, $\phi = 0.95$, $\rho = 4$	192.33	61	-163.83	221.33
7	$j_0 = 0.95$, $\phi = 0.05$, $\rho = 4$	348.33	238.33	-151.55	31.33
8	$j_0 = 0.95$, $\phi = 0.99$, $\rho = 8$	76.33	10.7	-224.88	606

The first value changed is the forgiveness factor ϕ. Lowering this from 0.95 to 0.45 shows an average of 30 more collisions at high speed, but a better average return. It could be that a lower ϕ lets the agent approach the behavior of flat RL which might explain the higher number of high speed collisions. However when looking at position 3 in the table, which lowers ϕ even further, a pattern only seems to emerge in the amount of total collisions which decreases and in the average return which approaches 0. Settings of position 3 would also explain the low amount of interjections as this agent performs more like a flat RL agent with most states having a very low overruling probability in only a few visits.

Position 4 looks at the difference of changing the initial probability. Comparing it to position 1, the return is hardly affected though the amount of interjections is higher. It is likely that a lower initial probability results in more collisions when approaching an obstacle the first time. This would cause the judge to quickly increase the overruling probability with the punishment factor and thus interjecting more.

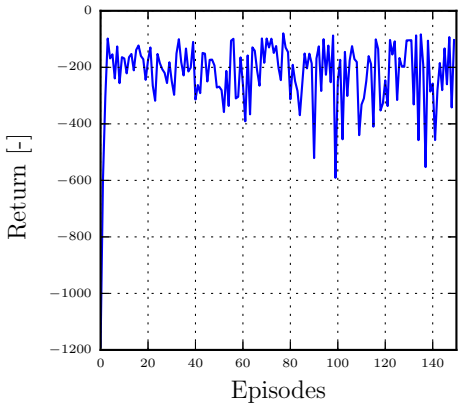
Next the punishment factor ρ is increased to 2 in position 5. The effect this has compared to position 1 seems to be a reduction in amount of collisions. The total collisions went from 250 to 177 while the percentage of high speed collisions to total went from roughly 60% to about 20%. This factor seems to have a strong effect on the safety of the agent. In addition the average return seems to also have improved so the agent has performed better in both safety and in its search for the optimal performance. Unfortunately the benefits of increasing ρ cannot always be relied upon as shown in position 6 where $\rho = 4$. Although the ratio of high speed to total collisions is still better than in the baseline ($\sim$30% compared to $\sim$60%), it is worse than in position 4.

Positions 7 and 8 experiment with combinations of ρ and ϕ. Again lowering ϕ shows an increase in collisions and a low amount of interjections and ρ might not be high enough to counter this. After all, a state is 'forgiven' whenever nothing negative happens and in the experiment performed, this occurs a lot more than the amount of collisions which result in a state 'punishment'. Maintaining a ϕ close to 1 and a high ρ results in the safest performance of the tests. Only 10.7 collisions at high speed for the judge settings as in position 8 which is roughly 14% of the total. However the performance in terms of average return is not as good as that of position 1, but this is an understandable outcome considering the high amount of interjections, resulting in more actions being needed to reach the goal and thus a lower return.

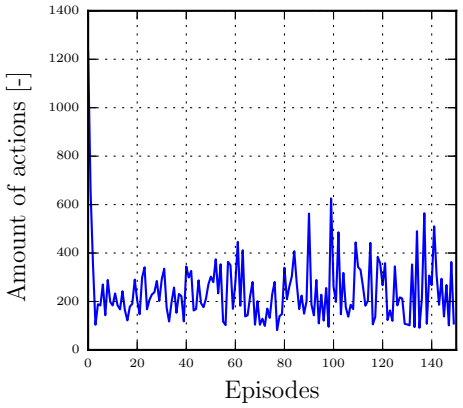
Although not every aspect of this experiment shows a consistent pattern, some conclusions can still be made, namely that the performance depends on both ϕ and ρ dependently. Having high values for both these factors results in less collisions albeit less optimal behavior but this becomes then a trade off between safety and optimality. The settings of these numbers is also quite dependent on the task as the occurrence of unsafe states becomes a factor in how fast or slow the overruling probability changes. For a given task, a conservative approach might be to start with a j_0 and ϕ close (less and not equal) to 1 and a high ρ.

As a demonstration of these settings, the best trial is shown in Figure 14 to compare with Figure 12. In this trial, there have been only 19 collisions of which 5 occurred at speeds over 2.5 m/s. Compared to the

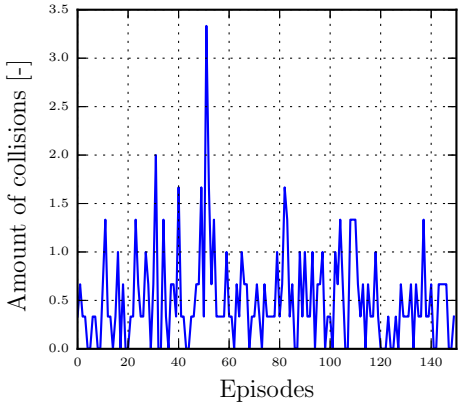
flat RL learner, the amount of collisions have therefore been reduced by 99.5 % in this particular run. Of course this is the best trial of the three, but even compared to the average of 76 collisions, the total collisions have been decreased by over 98% showing clear improvement.



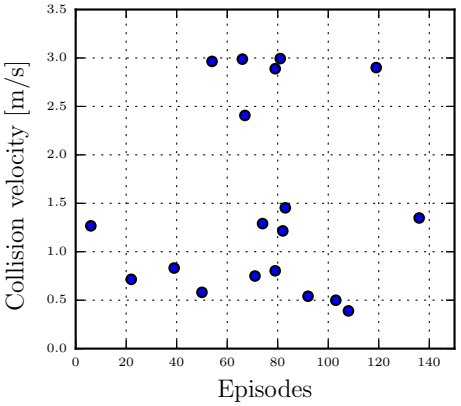
(a) Average reward



(b) Average number of actions



(c) Average number of collisions



(d) Run with the least amount of collisions, showing the velocity of each collision

Figure 14: Performance over 3 runs of the HRL agent performing the task augmented with the judge No. 8 of table 3

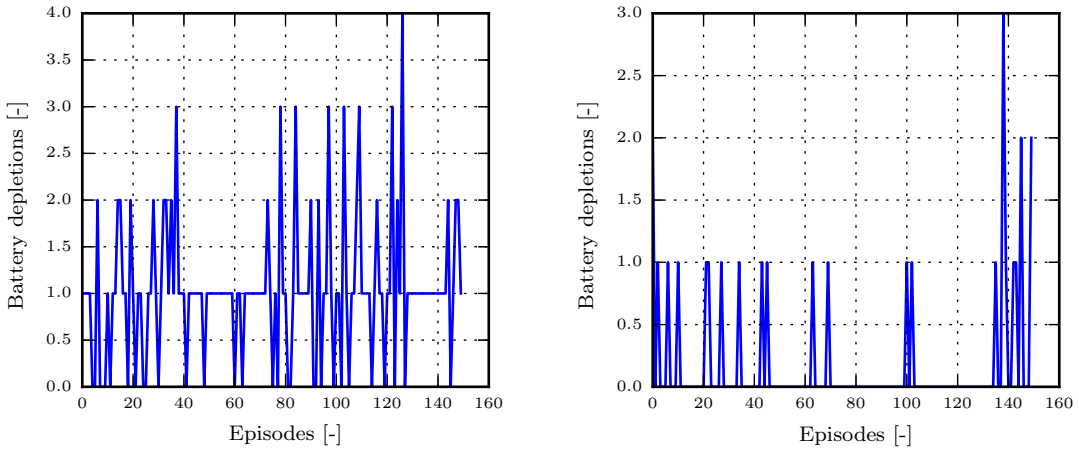
G. Modular implementation of the actor-judge method

In the interest of demonstrating the modularity of the actor-judge method, the SHRL agent is further augmented with another judge at the root level concerned with either recharging the battery or exploring the state space. An unsafe state in this level in the hierarchy is being low on battery. The HRL agent at the root level must decide to either use a fixed policy to recharge its battery or explore its state domain. Choosing the explore strategist, the strategist will select a sequence of up to 5 tacticians to either find the goal or not and hand control back to the root in case of the latter. The judge in this case will overrule the choice of the root with recharging the battery as in this case it's the safe choice.

It is outside the scope to find the optimal settings for this particular judge as the sensitivity of the judge initial settings and subsequent contribution to safety has already been explored. It is assumed at this point that the same kind of sensitivity can be found at different levels.

The results shown here in Figure 15 serve to demonstrate the possibility and give a brief insight on the effect a judge can have when implemented in a HRL learning machine. Comparing Figure 15b with Figure 15a, it is shown that the amount of battery depletions has decreased when the judge can overrule the agent.

Note that an episode is only ended when the goal is reached, a battery is considered 'depleted' when it reaches a lower bound of 10%; the agent will at this point be allowed to recharge and continue the episode which explains multiple depletions in a single episode. What the results here show is that the actor-judge method can be applied on a HRL agent and it will have a beneficial effect.



(a) Amount of battery depletions without a judge intervening

(b) Amount of battery depletions with a judge intervening

Figure 15: Comparison between judge intervening or not on a higher level in the hierarchy

VI. Conclusion

This research has a multitude of areas to investigate but mainly, it has been agent-safety oriented. Reinforcement Learning (RL) inherently has no additional method in place to avoid states dangerous to agent integrity such a Unmanned Aerial Vehicle (UAV) avoiding to collision against obstacles. This is largely due to RL approaches needing to explore its state space (including unsafe areas) in order to learn what is optimal and safety can largely be considered a side effect.

In this research, safety is the top priority and the effect of hierarchies to RL on safety is investigated using a path-finding task in a simulated environment where wall collisions are to be avoided. This research focuses on a simulated UAV exploring the unknown environment and is given range sensor information and a speedometer to determine its state relative to its surroundings.

A Hierarchical Reinforcement Learning (HRL) agent has a smaller state space to explore than a flat RL agent and therefore this increases the safety of an agent. This is due to previously learned behavior that can be reapplied in other areas of the environment. Results show that HRL has a clear benefit over flat RL in terms of safety which seems to indicate that HRL is already a valid approach to Safe Reinforcement Learning (SRL) even without additional safety backups.

In addition to investigating the benefit of HRL, an algorithm is developed that brings additional safety considerations to the HRL agent. This algorithm has is named the *actor-judge method* as it consists of the RL agent and a so-called judge which acts like a filter on the actions proposed by the agent. The judge is initialized as a system that has a probability to override the agent's actions. This probability to override is decreased or increased depending on the occurrence of negative events in the next state using a *forgiveness factor* and *punishment factor* respectively.

Although the actor-judge method does not fully eliminate the occurrence of what could be destructive states in physical systems (a UAV colliding full speed into a wall), it can significantly reduce the amount of times this may happen. Both the total amount of collisions are decreased as well as the number of high-speed collisions.

The actor-judge method is not developed as a means to immediately reduce the number of fatal state occurrences to 0 because, without supplying the agent with additional information i.e. policies or outside

information, it is unlikely to expect a model-free, autonomous, from-experience-learning agent to always entirely avoid unsafe states. However it is possible that this simple and modular approach can be rather easily be combined with other forms of safety-concerned approaches to reduce the number of unsafe actions even further.

References

[1]Guo, Q., Zuo, L., Zheng, R., and Xu, X., *A Hierarchical Path Planning Approach Based on Reinforcement Learning for Mobile Robots*, Springer Berlin Heidelberg, Berlin, Heidelberg, 2013, pp. 393–400.

[2]Wei, C., Zhang, Z., Qiao, W., and Qu, L., "An Adaptive Network-Based Reinforcement Learning Method for MPPT Control of PMSG Wind Energy Conversion Systems," *IEEE Transactions on Power Electronics*, Vol. 31, No. 11, Nov 2016, pp. 7837–7848.

[3]Bou-Ammar, H., Voos, H., and Ertel, W., "Controller design for quadrotor UAVs using reinforcement learning," *2010 IEEE International Conference on Control Applications*, Sept 2010, pp. 2130–2135.

[4]Junell, J., van Kampen, E., de Visser, C., and Chu, Q., "Reinforcement Learning Applied to a Quadrotor Guidance Law in Autonomous Flight," *AIAA Guidance, Navigation, and Control Conference*, 2015.

[5]Sutton, R. S. and Barto, A. G., *Reinforcement Learning: An Introduction*, MIT Press, 1998.

[6]Watkins, C. J. and Dayan, P., "Q-Learning," *Machine Learning*, Vol. 8, 1992, pp. 279–292.

[7]Minsky, M., "Steps toward Artificial Intelligence," *Proceedings of the IRE*, Vol. 49, 1961, pp. 8–30.

[8]Michie, D. and Chambers, R. A., "BOXES: An Experiment in Adaptive Control," *Machine Intelligence*, edited by E. Dale and D. Michie, Oliver and Boyd, Edinburgh, UK, 1968.

[9]Barto, A. G. and Mahadevan, S., "Recent Advances in Hierarchical Reinforcement Learning," *Discrete Event Dynamic Systems: Theory and Applications*, Vol. 13, 2003, pp. 41–77.

[10]Parr, R. and Russell, S., "Reinforcement Learning with Hierarchies of Machines," *Proceedings of the 1997 Conference on Advances in Neural Information Processing Systems 10*, MIT Press, 1998, pp. 1043–1049.

[11]Sutton, R. S., Precup, D., and Singh, S., "Between MDPs and semi-MDPs: A framework for temporal abstraction in reinforcement learning," *Artificial Intelligence*, Vol. 112, 1999, pp. 181 – 211.

[12]Dietterich, T. G., "The MAXQ Method for Hierarchical Reinforcement Learning," *In Proceedings of the Fifteenth International Conference on Machine Learning*, Morgan Kaufmann, 1998, pp. 118–126.

[13]García, J. and Fernández, F., "A Comprehensive Survey on Safe Reinforcement Learning," *Journal of Machine Learning Research*, Vol. 16, 2015, pp. 1437–1480.

[14]Mannucci, T., van Kampen, E., de Visser, C., and Chu, Q., "Hierarchically Structured Controllers for Safe UAV Reinforcement Learning Applications," *AIAA Information Systems-AIAA Infotech@ Aerospace*, 2017.

[15]Mannucci, T. and van Kampen, E., "A hierarchical maze navigation algorithm with Reinforcement Learning and mapping," *2016 IEEE Symposium Series on Computational Intelligence (SSCI)*, Dec 2016, pp. 1–8.

[16]Mannucci, T., van Kampen, E., de Visser, C., and Chu, Q., "Safe Exploration Algorithms for Reinforcement Learning Controllers," *IEEE Transactions on Neural Networks and Learning Systems*, Vol. PP, No. 99, 2017, pp. 1–13.

[17]Mannucci, T., van Kampen, E., de Visser, C., and Chu, Q., "SHERPA: a safe exploration algorithm for Reinforcement Learning controllers," *AIAA Guidance, Navigation, and Control Conference*, 2015.

[18]Cetina, V. U., "Autonomous Agent Learning Using an Actor-critic Algorithm and Behavior Models," *Proceedings of the 7th International Joint Conference on Autonomous Agents and Multiagent Systems - Volume 3*, AAMAS '08, International Foundation for Autonomous Agents and Multiagent Systems, Richland, SC, 2008, pp. 1353–1356.