Learning to Fly by Combining Reinforcement Learning with Behavioural Cloning

Eduardo F. Morales EDUARDO.MORALES@ITESM.MX
Departemento de Computación, Tec de Monterrey - Campus Cuernavaca, Temixco, Morelos 62589, México

Claude Sammut CLAUDE@CSE.UNSW.EDU.AU
ARC Centre of Excellence for Autonomous Systems, University of New South Wales, Sydney NSW 2052, Australia

Abstract

Reinforcement learning deals with learning optimal or near optimal policies while interacting with the environment. Application domains with many continuous variables are difficult to solve with existing reinforcement learning methods due to the large search space. In this paper, we use a relational representation to define powerful abstractions that allow us to incorporate domain knowledge and re-use previously learned policies in other similar problems. We also describe how to learn useful actions from human traces using a behavioural cloning approach combined with an exploration phase. Since several conflicting actions may be induced for the same abstract state, reinforcement learning is used to learn an optimal policy over this reduced space. It is shown experimentally how a combination of behavioural cloning and reinforcement learning using a relational representation is powerful enough to learn how to fly an aircraft through different points in space and different turbulence conditions.

1. Introduction

Suppose that we want to learn how to control an aircraft in a high-fidelity simulator. This task can be easily formulated as a reinforcement learning problem, where we want to learn which action to perform in any particular state to maximize the total expected reward reflecting a successful flight. However, this problem, whether the aircraft is real or simulated, typically has 20 to 30 variables, most of them continuous, describing an aircraft moving in a potentially "infinite" three dimensional space, hampering the applicability of reinforcement learning.

Several approaches have been suggested to deal with large search spaces, such as state abstraction, function approximation, and hierarchical decomposition. In this paper, states are abstracted using a relational representation (see (Morales, 2003)). This has several advantages: it is easy to represent powerful abstractions, simplifying the learning process; domain knowledge can be easily incorporated into the learning task; and once a policy has been learned, it can be re-used, without any further learning in another similar domain. Our state abstraction requires the definition of an adequate set of relations and a set of actions that operate on these relations. We call these sets of actions *r-actions*. In this paper, it is shown how *r-actions* can be automatically induced from traces of flights using behavioural cloning. Since turbulence is added to flights, the pilot is not always consistent in his/her actions and due to the characteristics of our particular behavioural cloning algorithm, a single state can have several applicable *r-actions*. Our behavioural cloning approach, however, induces only a small subset of possible *r-actions* per state, from which reinforcement learning obtains an optimal policy in a small number of episodes. The policy learned to fly the aircraft is tested under different turbulence conditions and it is shown that it can be used to fly a completely different mission including new manoeuvres not presented in the original traces without any further learning.

Section 2 describes the characteristics of the flight simulator. Section 3 reviews reinforcement learning using a relational representation. Section 4 describes the behavioural cloning approach used in the experiments. Section 5 provides experimental results with

Appearing in *Proceedings of the 21^{st} International Conference on Machine Learning*, Banff, Canada, 2004.

the flight simulator. Section 6 reviews some relevant related work. Finally, in section 7, conclusions and future research directions are given.

2. The flight simulator

A flight simulator based on a high fidelity model of a high performance aircraft (a Pilatus PC-9 acrobatic air plane) was used in our experiments. The PC-9 is an extremely fast and manoeuvrable aircraft used for pilot training. The model, provided by the Australian Defense Science and Technology Organization (DSTO), is based on wind tunnel and in-flight performance data.

The aircraft can be controlled with the ailerons, elevators, throttle, flaps and gear levers. The ailerons control the roll and yaw of the aircraft. The elevators control the pitch, the throttle controls the thrust of the plane, the flaps are used during landing and takeoff to increase lift and the gear is retracted during the flight. Since the flight simulator is of an aerobatic aircraft, small changes in control can result in large deviations in the aircraft position and orientation. This paper only deals with controlling the ailerons and elevators, which are the two most difficult tasks to learn. In all the experiments, it was assumed that the aircraft was already in the air with a constant throttle, flat flaps and retracted gear. Turbulence was added during the learning process (both behavioural cloning and reinforcement learning) as a random offset to the velocity components of the aircraft, with a maximum displacement of $10 ft/s$ in the vertical direction and $5 ft/s$ in the horizontal direction.[1]

The flight simulator produces a symbolic output given as Prolog facts. It includes information about the position of the aircraft, its velocity and orientation, roll, pitch and yaw rates, and the position of objects, such as buildings, that appear in the visual field. A flight is specified by a sequence of ways points. The aircraft is required to fly through each way point with a tolerance of $100 ft$ vertically and horizontally. Thus the aircraft must fly through a "window" centred at the way point.

3. Reinforcement Learning with relational abstractions

Trying to directly learn in the search space produced by the output information of the flight simulator is challenging for any reinforcement learning approach. Even with a very gross discretization of the space it is easy to produce millions of state-action pairs[2]. Also, once a policy has been learned to achieve a particular goal or sets of goal, a new policy has to be learned again if the goal changes its location. What we want to learn is a single policy that can be used to fly from any initial state to any achievable goal position in space under different turbulence conditions. In this research we use a relational representation to achieve this, where it is easy to encode the relative position of an agent with respect to a goal or to other objects in the environment. The main idea is to represent states as sets of properties that can be used to characterize a particular state and which may be common to other states.

An r-state *is a conjunction of first-order predicates.* The extension of an r-state is the set of states that are covered by its description. Each state is an instance of one and only one *r-state*. In the flight simulator, these properties could represent the relative distance to the target, the relative orientation of the aircraft to the target, the current plane roll, etc. For example, an *r-state*, such as *distance_target*(*State, close*) and *orientation_target*(*State,left*), covers all the states where the current target is close to and to the left of the aircraft. In this paper it is assumed that the relevant relations to characterize the space are previously defined by the user.

When learning a policy by reinforcement learning, we would like to only try those actions that are *relevant* to the current state. *An* r-action *is defined by a set of pre-conditions and a* generalized *action.* The preconditions consist of a conjunction of predicates that must hold for the *r-action* to be applicable. The *generalized* action represents all the instantiations of primitive actions that satisfy the conditions. When several primitive actions satisfy the conditions of an *r-action*, one of them is chosen randomly. An *r-action* can be applied to many states, and not all the *r-actions* apply to all the *r-states*.

For example, the following *r-action* with id number 23 (first argument), for controlling the aileron (second argument) says, if the aircraft is near its goal, the goal is to the left, the aircraft is rolled to the right and the roll rate is increasing, then move the stick to the far left.

[1]Although this is not a strictly accurate model of turbulence, it is a reasonable approximation for these experiments.

[2]For instance, consider a small space of 10 km^2 with 250m. of height. A gross discretization of $500 \times 500 \times 50$ m. with 5 possible values for yaw, pitch and roll, and 5 possible actions per state, gives us 1,250,000 state-action pairs.

Table 1. The rQ-learning algorithm.

Initialize $Q(S, A)$ arbitrarily
(where S is an *r-state* and A is an *r-action*)
Repeat (for each episode):
  Initialize s
  $S \leftarrow rels(s)$ % set of relations on state s
  Repeat (for each step of episode):
    Choose A from S using a *persistently exciting* policy (e.g., ϵ-greedy)
    Randomly choose action a applicable in A
    Take action a, observe r, s'
    $S' \leftarrow rels(s')$
    $Q(S, A) \leftarrow Q(S, A) + \alpha(r + \gamma max_{A'} Q(S', A') - Q(S, A))$
    $S \leftarrow S'$
  until s is terminal

r_action(23,aileron,State,move_stick(farleft)) $\leftarrow$
  distance_goal(State, close) $\wedge$
  orientation_goal(State, left) $\wedge$
  plane_rol(State, right) $\wedge$
  plane_rol_trend(State, inc) $\wedge$
  move_stick(farleft).

A policy consistent with our representation, which we will refer to as an *r-space* policy (π_R), is a scheme for deciding which *r-action* to select when entering an *r-state*. An *r-space* optimal policy (π_R^*) is a policy that achieves the highest cumulative reward among all *r-space* policies.

In the experiments reported below, we use Q-learning in *r-space*. However, other reinforcement learning algorithms may also be used. It is also easy to incorporate eligibility traces and function approximation into our approach. Table 1 gives the pseudo-code for the rQ-learning algorithm. This is very similar to the Q-learning algorithm, except that the states and actions are characterized by relations. The algorithm still takes primitive actions (a's) and moves over primitive states (s's), but learns over *r-state-r-action* pairs. This process is, in general, *non* Markovian, nevertheless, it can be shown to converge to an optimal *r-space* policy, using Singh's results (Singh et al., 1996) (see (Morales, 2003) for a more complete description).

The aircraft is controlled by performing left-right and forward-backward movements on the stick. We decided to divide the task into two independent reinforcement learning tasks: (i) forward-backward movements to control the elevation of the aircraft and (ii) left-right movements to control the roll and heading of the aircraft. We assume that in normal flight, the aircraft is approximately level so that the elevators have their greatest effect on elevation and the ailerons on roll.

To characterize the states for elevation control the following predicates and discretized values were defined:

- *distance_goal*: relative distance between the plane and the current goal. Possible values: *close* (less than 100 ft), *near* (between 100 and 1,000 ft), and *far* (more than 1,000 ft).
- *elevation_goal*: difference between current elevation of the aircraft and the goal elevation, considering the plane's current inclination. Possible values: *far_up* (more than 30^o), *up* (between 5^o and 30^o), *in_front* (between 5^o and -5^o), *down* (between -5^o and -10^o), and *far_down* (less than -30^o).

For aileron control, in addition to *distance_goal*, the following predicates and discretized values were also defined:

- *orientation_goal*: relative difference between current yaw of the aircraft and goal yaw, considering the current orientation of the plane. Possible values: *far_left* (less than -30^o), *left* (between -5^o and -30^o), *in_front* (between -5^o and 5^o), *right* (between 5^o and 30^o), and *far_right* (more than 30^o).
- *plane_rol*: current inclination of the plane. Possible values: *far_left* (less than -30^o), *left* (between -5^o and -30^o), *horizontal* (between -5^o and 5^o), *right* (between 5^o and 30^o), and *far_right* (more than 30^o).
- *plane_rol_trend*: current trend in the inclination of the plane. Possible values: *inc* (more than $+1$), *std* (between $+1$ and -1), *dec* (less than-1).

The ranges of the discretized values were chosen arbitrarily at the beginning of the experiments and defined consistently across different variables with no further tuning. The exact values appear not to be too relevant, but further tests are needed.

The above definitions discretize our state space. In simple domains, a suitable set of *r-actions* to perform on each *r-state* is relatively easy to define by hand. However, in more complex domains this is much more difficult and it is desirable to learn the *r-actions*. In this paper the *r-actions* were learned from traces of human pilots using behavioural cloning.

4. Behavioural Cloning

Behavioural cloning induces control rules from traces of skilled operators, e.g., (Sammut et al., 1992). The general idea is that if the human is capable of performing a task, rather than asking him/her to explain how it is performed, he/she is asked to perform it. Machine learning is then used to produce a symbolic description of the skill.

Flying an aircraft has been a benchmark problem in behavioural cloning. One of the main limitations of the original behavioural cloning approaches was that the clones produced were not robust to variations (Bratko et al., 1998). Several improvements have been made in recent years to produce more robust clones, by adopting a hierarchical decomposition of the learning problem and adopting different machine learning techniques. Some of the differences between our research and previous work are that we learn under high turbulence conditions in a sophisticated flight model (as in (Isaac & Sammut, 2003)). Our behavioural cloning approach is incremental with an easy to interpret representation language and we use an exploration phase to provide robustness to the system. Furthermore, in this research, behavioural cloning is not used as a means to learn how to perform a control task, but as guidance for reinforcement learning by inducing a small subset of relevant actions per state.

A very simple behavioural cloning approach is used to gather relevant *r-actions*. From a log of a human flight, for each state description, the algorithm evaluates a set of predefined predicates, such as those given in the previous section, and observes the control action. It the control action is an instance of an already defined *r-action* that *r-action* is used. Otherwise a new *r-action* is created with the conjunction of the predefined predicates using the format described below. This scheme can also be used in an incremental way, where new traces can be incorporated at any time (possibly) increasing the current *r-action* set (see also (Morales, 1997) for a similar approach used in chess end-games).

The actions were discretized as follows. The X component of the stick can have the following values: *farleft* (if stick X component value is less than -0.1), *left* (if it is between -0.1 and -0.03), *nil* (between -0.03 and 0.03), *right* (between 0.03 and 0.1), and *farright* (greater than 0.1). For the Y component of the stick movements the following discretization was used: *fardown* (above 0.4), *down* (between 0.3 and 0.4), *nil* (between 0.2 and 0.3), *up* (between 0.1 and 0.2), and *farup* (below 0.1). These discretizations were based on the actions performed by human pilots.

Elevation *r-actions* have the following format:

r_action(Num,el,State,move_stick(StickMove)) ←
  distance_goal(State,DistGoal) ∧
  elevation_goal(State,ElevGoal) ∧
  move_stick(StickMove).

where *Num* is an identification number, *StickMove* is one of the possible values for stick on the Y coordinate, *DistGoal* is one of the possible values for *distance_goal*, and *ElevGoal* is one of the possible values for *elevation_goal*. For elevation there can be 75 possible *r-actions* (3 possible values for *DistGoal*, 5 for *ElevGoal*, and 5 for *StickMovement*).

Similarly, the format for the aileron *r-actions* is as follows:

r_action(Num,al,State,move_stick(StickMove)) ←
  distance_goal(State,DistGoal) ∧
  orientation_goal(State,Orient_Goal) ∧
  plane_rol(State,PlaneRol) ∧
  plane_rol_trend(State,RolTrend) ∧
  move_stick(StickMove).

where there can be 1,125 possible aileron *r-actions*.

A total of 222 *r-actions* (180 for aileron and 42 for elevation) were learned after 5 consecutive mission logs over the flight plan shown in figure 1.

4.1. Exploration Mode

The trace logs were produced with high turbulence. Since the pilot does not always behave consistently, several conflicting actions may be produced for the same state description. As we are learning only from seen cases, there may be some states descriptions not covered by the *r-actions* but which may occur in other flight manoeuvres. To compensate for this, the learned *r-actions* were used to fly the aircraft to try to reach previously unseen situations. In cases where there were several applicable *r-actions*, one was chosen randomly. Whenever the aircraft reached a new state description (where there was no applicable *r-action*), the system prompted the user for a suitable action, from which a new *r-action* was induced. Also the user was able to perform a different action in any state if he/she wished, even if there were some applicable *r-actions*. Exploration mode continued until almost no new *r-actions* were learned, which was after 20 consecutive exploratory flights. In total, 407 *r-actions* were learned, 359 for aileron (out of 1,125 which is ≈ 32%) and 48 for elevation (out of 75, which is ≈ 64%). So although, we are still learning a substantial number of *r-actions* behavioural cloning helps us to learn only

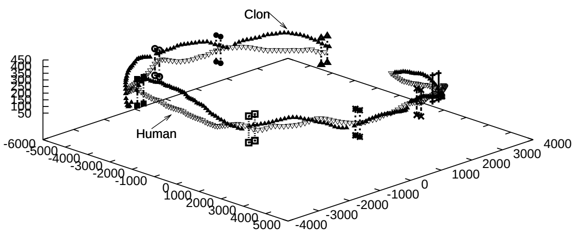

Figure 1. Human trace and the trace using the learned policy with reinforcement learning and behavioural cloning on the 8-goal mission with the maximum level of turbulence.

a subset of the possible *r-actions* (only one third) focusing the search space and simplifying the subsequent reinforcement learning task (an average of 1.6 *r-actions* per aileron state and 3.2 per elevation state).

The actual value of the stick position was assigned as the mid point of the intervals, except for the extreme ranges, as follows. For the X coordinate: *farleft*$= -0.15$, *left*$= -0.05$, *nil*$= -0.01$, *right*$= 0.05$, and *farright*$= 0.15$. For the Y coordinate: *fardown*$= 0.45$, *down*$= 0.35$, *nil*$= 0.25$, *up*$= 0.15$, and *farup*$= 0.05$. We have left for future work the use of continuous actions

Once the state has been abstracted and the *r-actions* induced, rQ-learning is used to learn a suitable policy to fly the aircraft.

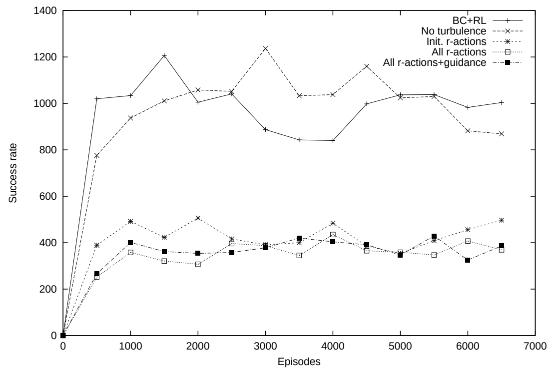

Figure 2. Learning curve for aileron for the different experimental set-ups while training.

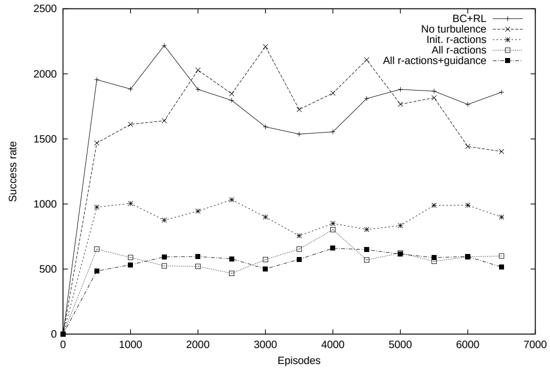

Figure 3. Learning curve for elevation for the different experimental set-ups while training.

5. Experiments

In all the experiments, the Q values were initialized to -1, $\epsilon = 0.1, \gamma = 0.9, \alpha = 0.1$, and $\lambda = 0.9$ (since we used eligibility traces). The experiments were performed on the 8 goals mission shown in figure 1. If the aircraft increases its distance to the current goal, after 20 time steps have elapsed from the previous goal, it is assumed that it has passed the goal and it changes to the next goal.

The following experiments were performed:

1. Positive reinforcement (+20) was given only when crossing a goal within 100 ft. with negative rewards otherwise (-1). In case the aircraft crashed or got into a state with no applicable *r-action*, a negative reward was given (-10). The experiments were performed with the maximum level of turbulence.
2. Same as (1) without turbulence.
3. Same as (1) but only with the *r-actions* learned from the original traces, i.e., without the exploration stage (222 *r-actions* in total).
4. Same as (1) but we automatically generate all the possible *r-actions* per state (5 with our discretization scheme) with 1200 *r-actions* in total.
5. Same as (4) but we use the original traces to "seed" Q-values, providing initial guidance.

Figures 2 and 3 show the learning curves of the above experiments for aileron and elevation respectively. In particular, how many times the aircraft crosses successfully the eight goals with maximum turbulence (every 500 flights) for aileron and elevation control. We continued the experiments for 20,000 episodes without any clear improvements in any of the experiments after the first 3,000 episodes.

As can be seen from the figures, without focusing the search with behavioural cloning, reinforcement learning is unable to learn an adequate strategy in a reasonable time. In complex environments, spurious actions

Table 2. Performance of the learned policy of experiment 1 after 1,500 episodes with different levels of turbulence on the eigth goals mission.

	Turbulence (m/s)/Tolerance					
Stage	0/ 100	0/ 200	5/ 100	5/ 200	10/ 100	10/ 200
Goal1	0	100	31	75	49	89
Goal2	100	100	16	41	26	46
Goal3	0	100	53	62	51	70
Goal4	0	0	23	35	27	46
Goal5	0	100	57	91	59	95
Goal6	0	0	16	33	24	47
Goal7	100	100	47	74	33	66
Goal8	0	100	35	58	45	61
Aver.	25	75.0	34.75	58.625	39.25	65.0

can very easily lead an agent to miss the goal. In this particular domain, going away from the current goal at some intermediate state can lead the agent into a situation where it is impossible to recover and reach the goal without first going away from the goal. The exploratory phase, where new *r-actions* were learned using random exploration, also proved to be useful as the initial traces substantially biased the learning process and limited its applicability.

The policy learned in experiment 1 after 1,500 flights was able to fly the whole mission successfully. Its robustness was tested under different turbulence conditions. Figure 1 shows a human trace of the mission and the trace followed by the learned policy. Table 2 shows the results, averaged over 100 trials, of flying the learned policy on the mission with different levels of turbulence. Two columns are shown per turbulence level, one with percentages passing the way point within 100 ft. (which was used in the reward scheme) and one with a 200 ft. The important point to note is that the aircraft can recover even if it misses one or more goals, and although it occasionally misses some of the goals, it gets quite close to them, as can be seen from figure 1.

Table 3 shows the average performance of the learned policies with and without turbulence (experiment 1 after 1500 episodes and experiment 2 after 3,000 episodes) using the *r-actions* learned with our behavioural cloning approach. It also compares the performance of a random selection of the *r-actions* learned with behavioural cloning and a random selection of all the possible *r-actions*.

The learned policies were then tested on a completely different mission, consisting of four way points. The intention was to try manoeuvres not previously seen before. The new mission included: a right turn[3], a sharper left climb turn of what it has previously seen before, another quick right turn, and a sharp descending right turn.

Table 3. Average performance of the learned policies with/without turbulence and performance of a random selection of the learned r-actions and of all the possible r-actions.

Turb./ Toler.	Policy w/turb.	Policy no/turb.	Rand. r-actions	Rand. all r-actions
5/100	34.75	30.25	16.25	1.125
5/200	58.625	56.00	38.00	1.875
10/100	39.25	26.75	16.50	0.625
10/200	65.00	50.50	38.875	2.00

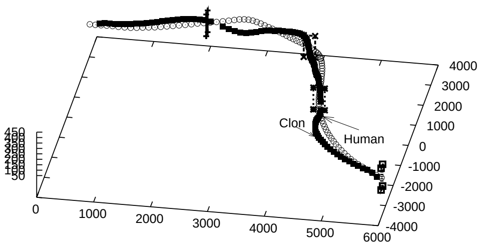

Figure 4. Flight path trace for a human and for the learned policy on a new mission with 4 goals.

Figure 4 shows a human trace and the trace using the previously learned policy (experiment 1) on the new mission with the maximum level of turbulence. The learned policy of the previous mission is clearly able to fly the aircraft on a completely new mission. Table 4 shows the performance of the policy on this new mission with different turbulence levels averaged over 100 trials. Table 5 shows the average performance of the learned policies with and without turbulence using the *r-actions* learned with our behavioural cloning approach on the new mission. It also compares the performance of a random selection of the *r-actions* learned with behavioural cloning and a random selection of all the possible *r-actions*.

6. Related Work

State aggregation clusters "similar" states together and assigns them the same value, effectively reducing

[3]The training mission involved only left turns.

Table 4. Performance of the learned policy on a different mission with different levels of turbulence.

Stage	Turbulence (m/s)/Tolerance 0/ 100	0/ 200	5/ 100	5/ 200	10/ 100	10/ 200
Goal1	0	100	66	99	74	100
Goal2	100	100	17	39	29	44
Goal3	100	100	38	70	46	70
Goal4	0	100	51	77	39	58
Aver.	50	100	43	71.25	47	68

Table 5. Average performance of the learned policies with/without turbulence and performance of a random selection of the learned r-actions and of all the possible r-actions.

Turb./ Toler.	Policy w/turb.	Policy no/turb.	Rand. r-actions	Rand. all r-actions
5/100	43.00	9.25	17.25	1.75
5/200	71.25	29.75	44.75	5.25
10/100	47.00	12.75	18.00	0.75
10/200	68.00	21.25	46.75	3.75

the state space. Work on tile-coding, Kanerva coding, and soft-state aggregation are some of the representatives of this approach. We also do state aggregation, but we use a relational representation, where it is easy to define powerful abstractions, to incorporate domain knowledge, and the learned policies can be directly used to other similar domains without any further learning.

Relational Reinforcement Learning (RRL) (Džeroski et al., 2001) also uses a relational representation for states and actions, however the main focus has been on approximating value functions with a relational representation. We are not trying to approximate a value function with a relational representation but rather, use a relational representation to structure and abstract the search space and then approximate a value function over this abstracted space. This makes our learning task much simpler and allows us to re-use previously learned policies more effectively. Also, our combination with behavioural cloning simplifies the reinforcement learning task as it provides a bias towards useful actions.

Traces from humans or possibly some available policies have been used by other researchers to provide initial guidance to reinforcement learning (e.g., (Ryan, 1998; Smart & Kaelbling, 2000; Driessens & Džeroski, 2002)). Our traces are used to learn a subset of actions per state. We also tried seeding initial Q values with all the possible *r-actions* using the original human traces, but we did not observe any clear improvements.

In (Ryan, 1998; Ryan, 2002), the user manually encodes a set of teleo-reactive operators (or TOPs), described using a relational representation. These are used by a planning system to produce sequences of subgoals at a high abstraction level. In our case, we only provide a set of relations, and learn the *r-actions* from logs of human traces. When applied to the flight simulator, they reported very poor results: less than 30% of successful flights with a large proportion of crashes (46%) in their best performances.

In (Ng et al., 2004) a reinforcement learning approach is decribed that learns how to fly an autonomous helicopter. Data from human pilots is used to fit a model of the helicopter's dynamics using locally weigthed regression and a reinforcement learning algorithm based on a Monte Carlo approach which pre-samples all the random numbers used in the stochastic simulation to find a good policy estimate. A careful selection of state variables and potential-based shaping rewards were used to learn how to hover and perform particular maneuvers. Maneuvers need to be specified by a set of contiguous points to follow. The human performance is used quite differently in this work compared with ours. Ng and Bagnell use data from human flights to constrict a model of the plant whereas, we use the data to constrain reinforcement learning. Our symbolic state abstraction approach in conjunction with behavioural cloning gives us a much simpler state-action space, although our actions are discrete. Also our maneuvers are specified just with positions of way points in space.

In (Isaac & Sammut, 2003) a robust behavioural cloning approach is described to learn how to fly. It is, to our knowledge, the only other work that uses a high fidelity flight model with different turbulence levels. The approach assumes that a flight should have a constant climb rate and turn rate to achieve a goal and divides the task into two parts: (i) given a goal position learn the value of the turn and climb rates for the aircraft to achieve the goal, and (ii) given a turn/climb-rate learn which actions to perform to achieve that particular turn/climb-rate. Both parts are learned using model trees, and in particular, the second part induces PID controllers at the leaves. Similar to our approach, once their system learns to perform particular manoeuvres, it can be used to fly different missions. Besides the obvious differences in the machine learning approaches, one of their advantages is that they can produce smoother flights as they use regression models, whether our behaviour is more "jumpy" due to the discretization used in the actions of the stick. However, to produce an adequate clone, they require an expensive wrapper to set up adequate parameters for the learning algorithms. Our research shows that reinforcement learning can be used in a complex control task and achieve comparable performance to the state-of-the-art behavioural cloning approach.

7. Conclusions and Future Work

In this paper we have shown how reinforcement learning can be used to solve a complex control problem. The strategy used is to define relations to abstract the search space, use logs of human traces to learn a subset of relevant actions, and use reinforcement learning over this abstracted and reduced search space to learn an optimal policy. The learned policy is shown to perform reasonably well under different turbulence conditions and on different missions.

There are two key elements that contributed to the success of this research: (i) a relational representation to produce powerful abstractions and descriptions including the relative position of the agent and (ii) the use of a behavioural cloning approach with an exploration phase to learn a subset of the possible actions per state, substantially reducing the reinforcement learning task.

There are several future research directions that can be followed. An obvious initial step is perform a more careful selection of the discretization ranges for the state variables. Another possibility is to incorporate regression models into the *r-actions* to produce "smoother" flights. We would like to explore how to learn new relations using an ILP algorithm.

Acknowledgments

The authors would like to thank Andrew Isaac for his help and feedback during the development of this work. This work was developed while the first author was on sabbatical leave at UNSW and supported by a grant from Conacyt (Mexico) and the ARC Centre of Excellence for Autonomous Systems (CAS).

References

Bratko, I., Urbančič, T., & Sammut, C. (1998). Behavioural cloning: phenomena, results and problems. automated systems based on human skill. *IFAC Symposium*. Berlin.

Driessens, K., & Džeroski, S. (2002). Integrating experimentation and guidance in relational reinforcement learning. *Proc. of the nineteenth international conference on machine learning* (pp. 115–122). Morgan Kaufmann.

Džeroski, S., Raedt, L. D., & Driessens, K. (2001). Relational reinforcement learning. *Machine Learning, 43(2)*, 5–52.

Isaac, A., & Sammut, C. (2003). Goal-directed learning to fly. *Proc. of the Twentieth International Conference on Machine Learning* (pp. 258–265). AAAI Press.

Morales, E. (1997). On learning how to play. In van den H. Herik and J. Uiterwijk (Eds.), *Advances in computer chess 8*, 235–250. The Netherlands: Universiteit Maastricht.

Morales, E. (2003). Scaling up reinforcement learning with a relational representation. *Proc. of the Workshop on Adaptability in Multi-agent Systems (AORC-2003)* (pp. 15–26).

Ng, A. Y., Kim, H. J., Jordan, M. I., & Sastry, S. (2004). Autonomous helicopter flight via reinforcement learning. *Advances in Neural Information Processing Systems 16*. Cambridge, MA: MIT Press.

Ryan, M. (1998). Rl-tops: An architecture for modularity and re-use in reinforcement learning. *Proc. of the Fifteenth International Conference on Machine Learning* (pp. 481–487). San Francisco: Morgan Kaufmann.

Ryan, M. (2002). Using abstract models of behaviours to automatically generate reinforcement learning hierarchies. *Proc. of the Nineteenth International Conference on Machine Learning* (pp. 522–529). San Francisco: Morgan Kaufmann.

Sammut, C., Hurst, S., Kedzier, D., & Michie, D. (1992). Learning to fly. *Proc. of the Ninth International Conference on Machine Learning* (pp. 385–393). Morgan Kaufmann.

Singh, S., Jaakkola, T., & Jordan, M. (1996). Reinforcement learning with soft state aggregation. *Neural Information Processing Systems 7*. Cambridge, MA: MIT Press.

Smart, W., & Kaelbling, L. (2000). Practical reinforcement learning in continuous spaces. *Proc. of the International Conference on Machine Learning* (pp. 903–910). Morgan Kaufmann.