Using reinforcement learning to control life support systems

Theresa J. Klein, Devika Subramanian, David Kortenkamp, Scott Bell

January 2004

Abstract

Advanced life support systems have many interacting processes and limited resources. Controlling and optimizing advanced life support systems presents unique challenges that are addressed in this paper. In particular, advanced life support systems are nonlinear coupled dynamical systems and it is difficult for humans to take all interactions into account to design an effective control strategy. We have developed a controller using reinforcement learning [1], that actively explores the space of possible control strategies, guided by rewards from a user specified long term objective function. We evaluated this controller using a discrete event simulation of an advanced life support system. This simulation, called BioSim, has multiple, interacting life support modules including crew, food production, air revitalization, water recovery, solid waste incineration and power. These are implemented in a consumer/producer relationship in which certain modules produce resources that are consumed by other modules. Stores hold resources between modules. Control of this simulation is via adjusting flows of resources between modules and into/out of stores. This paper describes the results of using reinforcement learning to control the flow of resources in BioSim. Our technique discovered unobvious strategies for maximizing mission length. By exploiting non-linearities in the simulation dynamics, the learned controller outperforms a handwritten controller.

1 Introduction

Keeping human beings alive in space is a complex task, particularly given the constraints imposed by launch costs from the surface of the earth. Weight considerations necessitate small buffers and low margins on consumables, so control policies that allocate energy and resources optimally are essential for success. Thus, optimal recycling of air and water is of paramount importance. For this reason NASA is developing Advanced Life Support systems (ALSs) that are designed to optimize recycling capabilities for maximum mission length. Because of the interactions of the various recycling systems in a closed environment, ALSs can be characterized as coupled dynamical systems. ALSs exhibit emergent behavior that cannot be simply explained as linear combinations of subsystem behavior. In addition, due to the presence of adaptive biological elements such as plants and humans, their dynamics change over time. The resulting non-linearity in the interactions makes designing controllers difficult. One potential solution is to use machine learning techniques to search the space of possible ALS controllers and identify optimal control policies. We do this search in simulation allowing for rapid convergence on interesting results. This paper describes a specific machine learning technique called *reinforcement learning* and a specific ALS simulation called BioSim [8]. We apply reinformcement learning to BioSim and compare its performance to that of a hand-written controller.

2 Reinforcement Learning

2.1 Background

Reinforcement Learning is a form of semi-supervised machine learning for acquiring optimal control policies in which a program receives rewards while performing a specific task [7]. Rewards come from the environment and can be received either incrementally or at an end state. For each state that exists in the environment the reinforcement learning calculates the value of a particular state based on the expected rewards that may be achieved by moving through it, and formulates a control policy by choosing action likely to lead to the highest expected reward. The value function can be expressed by Bellman's equation:

$$V(s) = \Sigma P(s, s')[r(s) + V(s')]$$

Where P(s,s') is the probability of moving from state s to state s'.

The most easily implementable form of reinforcement learning is Q-learning [16]. In Q-learning, a Q-function maps a value to a particular action in a particular state, rather than to just the state. This eliminates the need for the controller to contain a model of the system dynamics. Assuming the distribution of possible future states for the action in that state is fixed with time, as the Q-learner explores, the value function will incorporate knowledge of the expected value of future states. The Q-function only needs to know the expected value of the future rewards for that action in that state, and need not be able to predict the value of the next state.

Q-values are computed in an on-line fashion. That is, as the reinforcement learner is performing its task, it receives a reward, feeds that reward back to past state-action pairs, usually with a discount factor, and then chooses the next action based on its current value estimates. In earlier forms of reinforcement learning, updates to the Q-value are only made when a definite reward is received, such as at the end of a simulation run. By contrast, bootstrapping methods such as TD(λ) [12] use the current Q-value estimate as an approximation of the expected reward, and update past states on that basis. This permits faster learning to occur. In addition, the updates to each Q-value are factored by some learning rate, which is decayed over many episodes in which the task is performed, allowing the Q-values to converge to an optimum [16].

Q-Learners also need good exploration policies. Because the Q-learner chooses actions on the basis of incomplete value estimates, it is possible for it to select a non-optimal action if it always strictly chooses the highest valued estimate. This is known as greedy selection and can often result in the convergence to a suboptimal Q-function. As a result, a balance between "exploration" and "exploitation" must be struck. A variety of mechanisms for exploration strategies have been studied [15]. One common method is what is called ϵ-greedy selection, in which a random action is chosen with probability ϵ.

The theory of Q-learning just described was developed under the assumption that states and actions would be discrete and the values would be simply assignable to them in a tabular format. Dealing with Q-value for continuous valued states and actions, as we need to, is an area of significant on-going research in reinforcement learning. This area can be broadly divided into two major sections, continuous valued function approximators and discretization combined with tabular lookup. Continuous valued function approximators, such as neural networks, have been used in several applications [3, 4, 13] and have many advantages, such as speed of computation, and

implicit generalization capability. Neural nets are capable of taking a continuous valued state and action and computing a continuous value function as a result. Unfortunately, neural networks have significant complications when applied in the context of reinforcement learning, and tend to suffer from overtraining and "ill-conditioning" [2].

By contrast, table-lookup representations are still the only format for which RL is known to converge to an optimal value function. However, to apply table-lookup methods, we must discritize the state-space, and must also store every visited state. Because the number of states grows exponentially as the dimensionality increases, this becomes prohibitively computationally expensive beyond a small number of dimensions. A great deal of on-going research in reinforcement learning is devoted to overcoming this dimensionality problem through both continuous methods such as neural networks, above, and various methods of discretization and function approximation [5, 14, 9, 6]. We do not deal with these issues in this paper but present it only so that the format of our reinforcement learner can be placed within the context of the current state of the field, and to illustrate the difficulties involved in adapting a reinforcement learner to a task such as this.

2.2 Usefulness to Control

Because of the ability of Q-learning to learn optimal control actions without knowledge of a model of the system, it has significant potential for use in controlling ALSs. As described earlier, due to the coupling between recycling modules and the non-linear nature of elements such as the crew and plants, the system as a whole produces non-linear behavior for which we have no model. This is also important for a second reason. Although BIOSIM contains complex models of plants and crew, which could be used to design traditional controllers, in a real-life situation there are many aspects of human and plant biological behavior which are unknown. As a result, Q-learning has the advantage of being better able to adapt to real-life situations where plant and crew behavior is by necessity unknown.

3 An Integrated Life Support Simulation

Over the last several years NASA has been developing an integrated advanced life support simulation [8]. The simulation was developed in accordance with NASA requirements and baseline assumptions for the design of an ALS [11, 10]. The simulation includes detailed, stochastic models of the crew, air, water, biomass (including plant growth chambers), power, food production and solid waste recycling. Each of these components interconnects with the rest of the simulation (see Figure 1).

3.1 Carbon Dioxide Difference Equation

As an example of the kinds of equations that underly the integrated simulation, we present the carbon dioxide difference equation in detail. This equation calculates the amount of CO_2 flowing into the CO_2 tank from all CO_2 production sources.

$$\begin{aligned} CO_2'(t+1) &= CO_2(t) - CRSCO_2In(a(t, CRS), H_2(t), CO_2(t)) \\ &\quad + CrewCO_2Out(t) - PlantsCO_2In(t) \end{aligned}$$

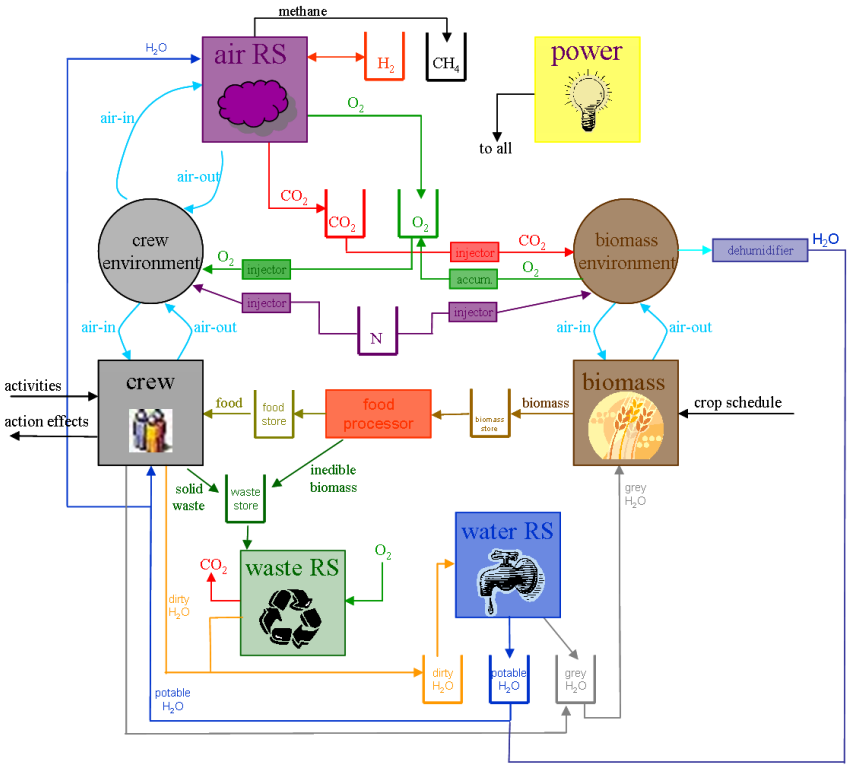

Figure 1: The modules that comprise the integrated advanced life support simulation and the connections between the models.

$CRSCO_2In(a(t,CRS),CO_2(t),H_2(t)) =$

$$\begin{cases} 0 & \text{if } H_2(t)=0 & \text{or } CO_2(t)=0 & \text{or } a(t,CRS)=false \\ 10 & \text{if } a(t,CRS)=true) & \text{and } (40 <= H_2(t) & \text{and } 10 \leq CO_2(t) \\ CO_2(t) & \text{if } a(t,CRS)=true) & \text{and } H_2(t) \geq 40 & \text{and } 0 < CO_2(t) \leq 10 \\ 0.25 * H_2(t) & \text{if } a(t,CRS)=true) & \text{and } 0 < H_2(t) \leq 40 & \text{and } CO_2(t) \geq 10 \\ min(CO_2(t), 0.25 * H_2(t)) & \text{if } a(t,CRS)=true) & \text{and } 0 < H_2(t) \leq 40 & \text{and } 0 < CO_2(t) \leq 10. \end{cases}$$

This equation gets rid of overflows of the tank capacity.

$$CO_2(t+1) = \begin{cases} 0 & \text{if } CO_2'(t+1) \leq 0 \\ CO_2'(t+1) & \text{if } 0 < CO_2'(t+1) < capacity(CO_2) \\ capacity(CO_2) & \text{if } CO_2'(t+1) \geq capacity(CO_2) \end{cases}$$

$CrewCO_2Out(t)$ is a function of time that depends on crew activity levels and the impulse response characteristics of the control loops controlling the cabin air composition. $PlantCO_2In(t)$ is a function of time that depends on the growth stage of the crop.

The state equation for the CO_2 tank depends upon the current levels in the Hydrogen tank, which is determined by the output of the OGS. Due to this coupling between the CRS and OGS, as well as the introduction of the non-linear elements of the plants and crew this simulation produces non-linear behavior. This is illustrated by Figure 2. Using only a linear model of the plants, we ran the simulator, and obtained results showing non-linear behavior. Although we only seek to control the WRS, OGS, and CRS in this paper, with even linear plants present in the system, the system we obtain a system for which we do not have a known model. This illustrates why reinforcement learning is potentially useful for this problem.

3.2 Implementing a reinforcement learner for the simulation

In order to assess the usefulness of standard reinforcement learning methods for controlling ALSs, we focus on the sub-task of controlling the air and water recycling subsystems. We assume that there is an adequate supply of power and food. The air recycling system has two active components which can operate independently, the OGS (Oxygen Generation System) and CRS (Carbon Dioxide Reduction System). We control each of these separately. Though the WRS (Water Recycling System) can also be broken down into smaller subunits, these components operate in series, therefore we treat the WRS as a single system.

We sense five quantities, the tank levels for carbon dioxide, oxygen, hydrogen, dirty water, and potable water, which are the storage areas for the inputs and outputs of the three controlled systems. Likewise, we control three quantities, the flow rates of dirty water, potable water, and carbon dioxide to the WRS, OGS, and CRS (the flow of hydrogen to the CRS is set passively by the CO_2 flow rate). The crew and plants are driving factors of the system, even though we do not control them directly. Figure 3 shows a diagram of the simplified model.

Although we are only controlling three input flow rates, the coupled dynamical characteristics of the system are preserved. For example, the CRS produces potable water, which is used by both the OGS, as well as the plants and the crew. The OGS in turn produces hydrogen, which is used by the CRS, and oxygen, which is consumed by the crew. The crew produces CO_2, which is consumed by the plants and the CRS. If either tank is near empty, the available hydrogen or carbon dioxide will limit the ability to recycle CO_2, which changes the rate at which potable water is added to

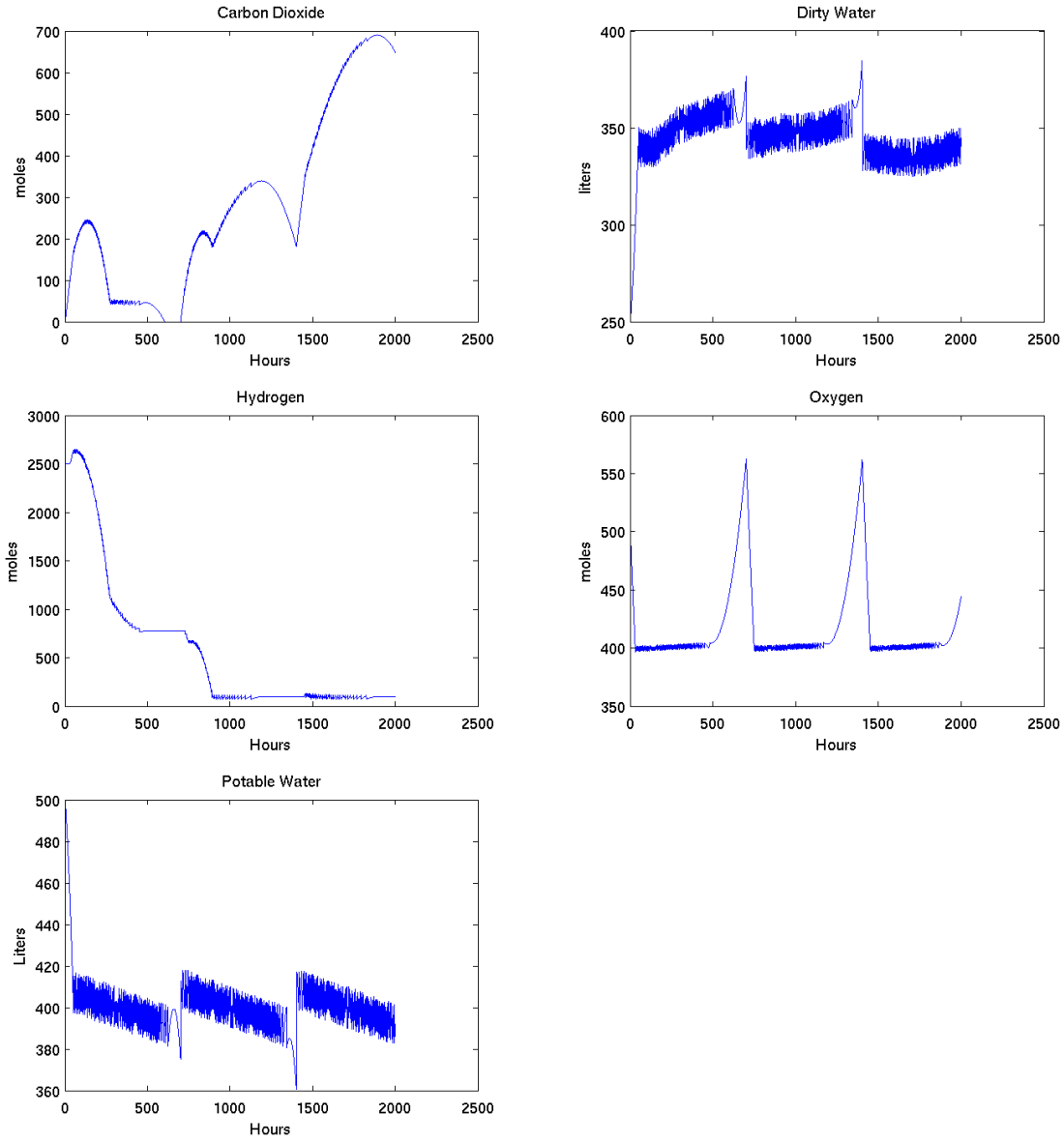

Figure 2: This plot shows an example of the simulator behavior if only a simple plant model is used. A linear change in the plants ability to recycle CO_2 causes non-linear behavior in the oxygen and carbon dioxide tank levels, as well as other state variables. This shows that a system with even simple linear components such as this is capable of producing complex behavior when the inputs and outputs are coupled.

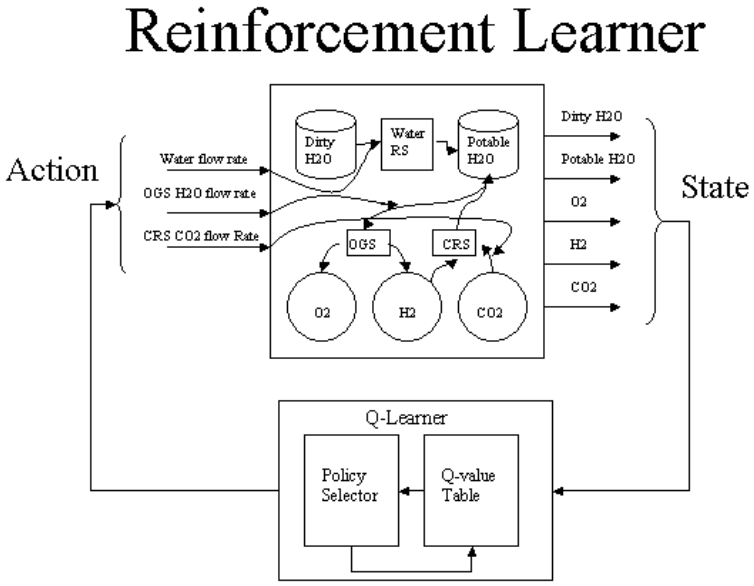

Figure 3: This diagram shows a simplified view of the system as it is controlled by our reinforcement learner. Tank levels for CO_2, H_2, O_2, dirty water and potable water are observed, and flow rates for dirty water WRS input, potable water OGS input, and CO_2 CRS input are controlled. The Q-learner acquires a value function for the state space, and selects control actions according to its current estimates.

the potable water tank, which can cause the potable water to be consumed more rapidly or more slowly in the production of oxygen.

Our reinforcement learner is a Q-learner using a table-lookup function approximator, ϵ-greedy selection, with the ϵ value decayed exponentially, and TD(λ) updates. We also have introduced a generalizing function that explicitly generalizes updates to surrounding states in the state lookup table. This is distinguished from the implicit generalization in function approximators such as neural networks. Our generalization method is still capable of being combined with table-based generalization methods such as interpolation [5], although we do not do this here.

Since we have five state variables and three action variables, a coarse grid of 10 points along each axis would result in a state table of 10^8 points. This an excessive amount of computational resources to devote to this problem. In order to reduce the size of the state space, we have reduced the actions to just two possibilities, an "on" flow rate and an "off" flow rate. This gets us down to just $10^5 * 2^3 = 800{,}000$ states, which is still very large. Since there are large portions of the state space that are unreachable (for example, the dirty water and potable water tanks cannot be full at the same time), we can reduce this further by including in the state table only the states which are actually visited. We do this in online fashion, by measuring the distance between each new state

and its nearest neighbor in the state table. If it is more than a fixed distance away from the nearest state (in our case 0.1), then it is added to the table. Otherwise the nearest point in the table is used. As a result we are able to achieve results using a state table of only a few tens of thousands of points, instead of the hundreds of thousands of "possible" states.

In the Table 1 we give a simplified outline of the reinforcement learning algorithm used in our controller.

4 Results

Our evaluation objective is to maximize mission length. The simulation is initialized with fixed amounts of resources in its tanks. The efficiency with which these resources are consumed is a determining factor in the length of the mission – as resources drop below critical levels the mission is ended. For the purposes of our experiment the initial stores of resources used by the OGS, CRS, and WRS are listed below, along with the tank capacities.

Resource	Initial	Capacity
Dirty Water	500L	500L
Potable Water	250L	500L
Carbon Dioxide	500mol	1000mol
Oxygen	500mol	1000mol
Hydrogen	2000mol	5000mol
Biomass	300kg	1000kg
Food	2000kg	2000kg

4.1 Baseline controller

To provide a baseline for comparison to the reinforcement learner, we developed a hand-written controller. The baseline controller operates by using fixed set points for turning on and off the subsystems. Each tank capacity has an upper and lower threshold. If one of the tanks is over its upper threshold the hand controller will turn off the subsystem that feeds it, and if it is below it's lower threshold the system will be turned off. Sometimes, combinations of different tank levels are used to determine on/off status. The thresholds are as follows:

Resource	Lower	Upper
Dirty Water	100L	400L
Potable Water	100L	400L
Carbon Dioxide	300mol	800mol
Oxygen	300mol	800mol
Hydrogen	1000mol	4000mol

Using this baseline controller and the initial resource levels mission last approximately 8000 ticks (a simulation tick represents one hour of mission time so this is roughly a year). However, because the baseline controller is based on straightforward thresholds that avoid the empty and full tank capacity levels, it is unable to exploit the non-linearities that exist in the system near these points. As we will see, by exploiting the non-linearities in these regions, reinforcement learning is able to discover conservation strategies that are more successful.

Definitions of parameters:
s : the current state - a five element vector for the tank levels of dirty water, potable water, carbon dioxide, hydrogen and oxygen
a : the current action a three element vector for the flow rates to the OGS, CRS and WRS
s': the next state reached from s through action a.
a': the action chosen by the ϵ -Greedy algorithm
a*: the optimal (according to current estimates) action for state s'
δ : the TD-λ update
γ : a discount for determining how much the next state feeds back to the current state
e(s,a) : the "eligibility" of (s,a) - determines how much of the update that state and action pair can receive
α : the learning rate
β : the decay factor for updating neighbors as a function of distance
λ : discount factor for decaying the eligibility of (s,a). The more often it is updated, the less of the update it accepts.

---

initialize eligibility trace e(s,a)
foreach episode
  Initialize s and a
  for each iteration
    $r = calculateReward(s, a)$ (get reward for transition from s through a)
    observe s'. (observe new state)
    $a' = eGreedySelect(s')$ (choose an action for state s')
    $a* = greedySelect(s)$ (find current best action for state s')
    $\delta = r + \gamma Q(s', a*) - Q(s, a)$ (compute TD update)
    $e(s, a) = e(s, a) + 1$ (set eligibility for (s,a) pair.)
    for all s,a in the trajectory,
      Get neighborhood of (s,a) - defined by N states within distance ϵ
      for all s", a" in the neighborhood of s,a
        d = Euclidean distance between (s,a) and (s",a")
        $Q(s'', a'') = Q(s'', a'') + \alpha\delta e(s, a)exp(-\beta d)$ (update neighboring states)
      end
      if (a'=a*) $e(s, a) = \gamma\lambda e(s, a)$ **else** e(s,a) = 0
    end
  end
end

Table 1: The basic Q-learning Algorithm used by our controller. Each iteration, the controller advances one step, observes its reward, and updates the previous state-action pair using TD-λ techniques combined with explicit generalization. It then chooses a new action based upon its current value estimates and current state.

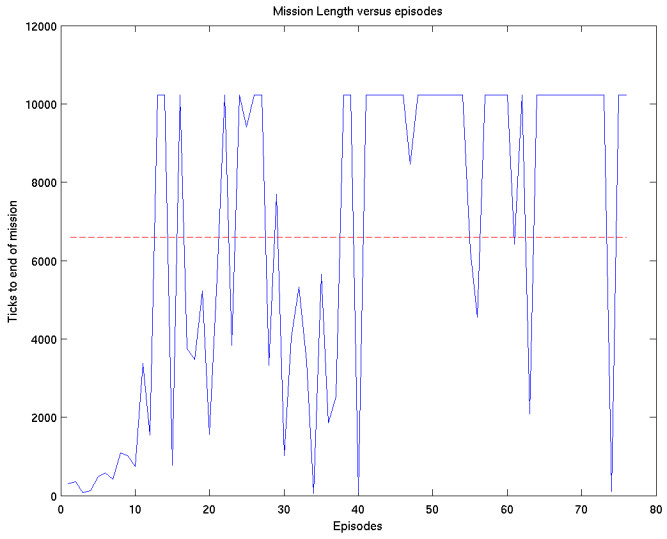

Figure 4: This figure shows the progress of the controller in discovering the water rationing trick that is used to extend mission length to the limit of the food supply at 10233 ticks, beating the hand controller's limit of approximately 6600 ticks, shown by the red line. It initially discovered the water rationing ability on the 13th episode. As time passes it achieves this limit with greater reliability.

4.2 Reinforcement Learner

The following plots, Figure 4 and 5, show the behavior of the reinforcement learning controller as it approached convergence. After only 13 episodes (trials), the controller has discovered a solution that allows it to achieve the maximum mission length of 10223 hours. Subsequently, the number of episodes in which it achieves the maximum mission length steadily increases.

The reinforcement learning controller quickly learns to conserve water by keeping the dirty water tank almost full and dispensing water only every dozen or so ticks, allowing the OGS to operate and the crew to drink. Also, the controller has also learned to recycle CO_2 as long as the hydrogen supply is available. Its behavior is not ideal, but it has effectively found a solution that maximizes the mission length and hence its reward. This method of water rationing actually exposes a weakness in the simulation that was not immediately apparent to its designers. The reinforcement learner has effectively found a technique that exploits the nonlinearity inherent in the design of the crew model to discover an unpredictable solution to the problem of extending mission length. This shows the potential that reinforcement learn has to solve complex problems such as those in ALS, as well as a bonus effect of exposing unobvious problems or features of the simulation.

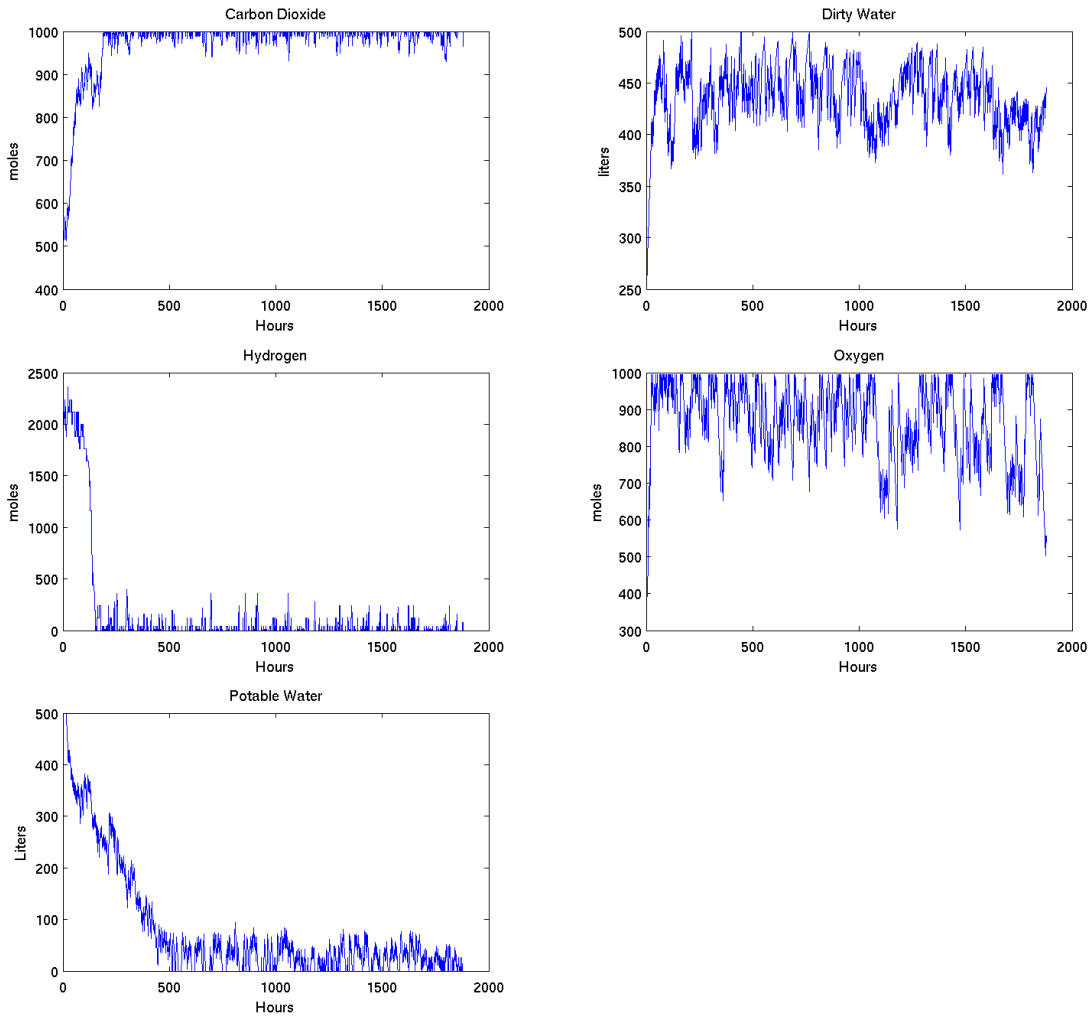

Figure 5: This figure shows the levels in the tanks through the course of the mission. After some initial mistakes, the controller is keeping most of the water in the dirty water tank for most of the mission, only recycling when necessary. Note: These results were obtained using the actual plant model incorporated into BIOSIM, which is very non-linear.

5 Conclusions and Future Work

We have demonstrated that reinforcement learning approaches can automatically find more optimal solutions than simple control approaches. We do not expect that entire control algorithms will be learned in this way, but we do believe that these approaches can outline broad solution classes that can then be investigated by control engineers. Expanding Q-learning of this type to more complex life support problems requires significant technical problems to be overcome, including methods of approximating the value function over high dimensions, and dealing with long trajectories and partial observability of the system state.

We plan to expand from our simple controllers to more complex problems. The non-linearity in the system is largely related to the interaction of the plants with the rest of the environment and the crew life has so far been limited mostly to the supply of food brought from Earth. In the long run, managing a large plant chamber that supplies a significant portion of the crew's food as well as the oxygen recycling capability promises to be a more challenging problem. We anticipate that some mixture of different crops would be desirable with staggered harvest and planting times. The challenge is to choose what plants to use, and what acreage, so as to maximize the mission life, which implicitly means optimizing oxygen recycling and food production at the same time. With this type of problem, we intend to explore a multi-agent policy-search, in which each agent controls one crop at a time, selects the crops and observes a global reward signal. This has the additional potential to allow us to determine what crop acreages and types are needed for optimal resource utilization.

References

[1] A. Barto and R. Sutton. *Reinforcement learning: an introduction*. MIT Press, 1998.

[2] Justin Boyan and Andrew Moore. Generalization in reinforcement learning: Safely approximating the value function. In *Advances in Neural Information Processing System 7*. The MIT Press, 1995.

[3] Rémi Coulom. *Reinforcement Learning Using Neural Networks, with Applications to Motor Control*. PhD thesis, Institut National Polytechnique de Grenoble, 2002.

[4] Robert H. Crites and Andrew G. Barto. Improving elevator performance using reinforcement learning. In David S. Touretzky, Michael C. Mozer, and Michael E. Hasselmo, editors, *Advances in Neural Information Processing Systems*, volume 8, pages 1017–1023. The MIT Press, 1996.

[5] Scott Davies. Multidimensional triangulation and interpolation for reinforcement learning. In Michael C. Mozer, Michael I. Jordan, and Thomas Petsche, editors, *Advances in Neural Information Processing Systems*, volume 9, page 1005. The MIT Press, 1997.

[6] Fernando Fernandez and Daniel Borrajo. On determinism handling while learning reduced state space representations. In *European Conference on Artificial Intelligence*, 2002.

[7] Leslie Pack Kaelbling, Michael L. Littman, and Andrew P. Moore. Reinforcement learning: A survey. *Journal of Artificial Intelligence Research*, 4:237–285, 1996.

[8] D. Kortencamp and S. Bell. Simulating advanced life support systems for integrated controls research. In *Proceedings of the third International Conference on Environmental Systems*. ICES, 2003.

[9] Remi Munos and Andrew W. Moore. Variable resolution discretization for high-accuracy solutions of optimal control problems. In *IJCAI*, pages 1348–1355, 1999.

[10] NASA-JSC. *Advanced Life Support Systems Baseline Values and Assumptions Document*.

[11] NASA-JSC. *Advanced Life Support Systems Requirements Document*.

[12] J. Peng and R. J. Williams. Incremental multi-step q-learning. In *Proceedings of the Eleventt International Conference on Machine Learning*, pages 226–232. International Conference on Machine Learning, 1994.

[13] D. E. Rummelhart and J. L. McClelland. *Parallel Distributed Processing: Explorations in the microstructure of cognition. Volume 1: Foundations*. The MIT Press, 1986.

[14] Richard S. Sutton. Generalization in reinforcement learning: Successful examples using sparse coarse coding. In David S. Touretzky, Michael C. Mozer, and Michael E. Hasselmo, editors, *Advances in Neural Information Processing Systems*, volume 8, pages 1038–1044. The MIT Press, 1996.

[15] S. B. Thrun. The role of exploration in learning control with neural networks. In D. A. White and D. A. Sofge, editors, *Handbook of Intelligent Control: Neural, Fuzzy and Adaptive Approaches*, Florence, Kentucky, 1992. Van Nostrand Reinhold.

[16] C.J.C.H. Watkins. *Learning from Delayed Rewards*. Phd. thesis, King's College, Cambridge, UK, 1989.