Command and Control of Teams of Autonomous Systems

Douglas S. Lange*, Phillip Verbancsics, Robert S. Gutzwiller,
John Reeder, and Cullen Sarles

Space and Naval Warfare Systems Center Pacific
San Diego, CA 92152
doug.lange@navy.mil

Abstract. The command and control of teams of autonomous vehicles provides a strong model of the control of cyber-physical systems in general. Using the definition of command and control for military systems, we can recognize the requirements for the operational control of many systems and see some of the problems that must be resolved. Among these problems are the need to distinguish between aberrant behaviors and optimal but quirky behaviors so that the human commander can determine if the behaviors conform to standards and align with mission goals. Similarly the commander must able to recognize when goals will not be met in order to reapportion assets available to the system. Robustness in the face of a highly variable environment can be met through machine learning, but must be done in a way that the tactics employed are recognizable as correct. Finally, because cyber-physical systems will involve decisions that must be made at great speed, we consider the use of the Rainbow framework for autonomics to provide rapid but robust command and control at pace.

Keywords: cyber-physical, command and control, autonomic, machine learning.

1 Introduction

Teams that include heterogeneous autonomous unmanned systems (AUS) are good generalizations of complex cyber-physical systems. They contain autonomous units connected by a network, involving distributed computation, and to further complicate matters may have cooperative intelligent behavior among changing subsets of the systems components. As the elements of these networked systems exist in and interact with the physical environment, the physical nature of AUS is obvious. Even the network can be influenced by the environment given the use of wireless communications.

Controlling such a complex system requires several critical capabilities. First, the goals and constraints for the AUS team must be communicated to the various decision making nodes. This may include all of the AUS, as they all may possess sufficient autonomous capability to decide how to act under many situations given the goals. A central controller, or more generally several distributed controllers, must have

* Corresponding author.

R. Calinescu and D. Garlan (Eds.): Monterey Workshop 2012, LNCS 7539, pp. 81–93, 2012.

confidence (particularly if human operated) that the goals and constraints have been received and correctly interpreted by the autonomous units. Second, the control units must have sufficient situational awareness of the environment and the behaviors of the team members in order to decide if changes to orders are required. The control must have the ability to determine if any error conditions are present and must be able to distinguish between aberrant behavior and what may be a plausible but unpredicted solution. Finally, team strategies must be selected to accomplish goals, and these strategies may need to be altered as the environment changes.

This paper explores the control of complex cyber-physical systems. In particular we look at the requirement for human controllers to influence the operation of these systems as well as addressing the need for autonomic control in situations where time constraints do not allow human decision making. This combination of requirements poses interesting demands on how cyber-physical systems are constructed and how they incorporate adaptation.

2 Control of Complex Systems

The control exercised by a military commander over forces is described as "…guiding the operation" [1]. The presumption is that there is a mission statement, a set of assets with which to perform the mission, and an environment to operate in that may include an opposing force. There are several ways in which a commander guides an operation.

Maintain alignment: The commander must ensure that all decisions remain aligned with the operation's mission and the commander's intent.

Provide situational awareness: The commander must assess the status of plan execution constantly, utilizing a common operational picture (COP).

Advance the plan: The commander must monitor the status of plan execution against the plan's timeline.

Comply with procedure: The commander oversees compliance with warfighting procedures to avoid mistakes (e.g., friendly fire engagements or collateral damage) and achieve efficiencies.

Counter the enemy: The commander must be responsive to emerging intelligence, surveillance, and reconnaissance information that differ significantly from expectations.

Adjust apportionment: Changes to asset availability or changes to requirements and priorities may require reapportionment of assets.

Military organizations are essentially complex cyber-physical systems. The end – nodes may be aircraft with pilots, or aircraft without pilots. These units whether manned or unmanned can be viewed as autonomous systems that cooperate to achieve a mission under the command of a human commander.

The tasks of the military commander are also clearly analogous to what would be required in many non-military systems. The only difference may be that no enemy exists, but the environment is nevertheless capable of surprising, therefore emerging information that alters assumptions is still possible. Units may become inoperable just as in military operations, and adjustments to plans must often be made.

As the complexity of the system increases, the commander must work at higher, more abstract levels. The units of the system must also exhibit higher levels of

autonomy so that decision making is moved further down and is more immediate [2]. Based on the level of autonomy exhibited by the units, we can model the size of an operation that a single person can manage, provided the situation can be adequately described to the commander.

3 Control of Systems by Humans

Automation can easily be called ubiquitous. We interface with it daily, even if we do not immediately recognize it. In years past for example, elevators required human operators, but now we simply press a button to reach our floor. Even highly complex systems integrate automation; commercially flown airplanes have autopilots that are capable of landing the plane, and some models of cars have automated systems which bypass the driver if safety is in doubt (i.e., automatic braking systems, vehicle headway monitoring). These systems integrate automation, but still rely heavily on a human component for routine performance and supervision. While automation is becoming more common, and more reliable, it rarely replaces or removes a human with experience from the overall task [3].

Automation has also been shown to result in phenomena such as complacency which results in operators failing to detect failures of automated systems [4,5,6] and automation bias, which results in operators blindly following automation recommendations or failing to act unless the automation requests the human action in decision making systems [7,8,9,10,11].

Leli and Filskov [12] suggest that it is specifically the integration that plays a large role in determining the effectiveness of a system outcome. In their work, automated diagnostic systems consistently outperformed clinicians when in isolation; however decision accuracy decreased as a direct result of integrated clinician use of the aid to diagnose psychological conditions. This suggests that perhaps the most critical aspect of automation is not the engineering behind the automation itself, but the interaction between any automation and the operator who is expected to work together with it.

Parasuraman and Wickens [3] also identified issues related to the ability of human operators to understand the actions of the automation. The operators trust and ability to evaluate the performance of autonomous systems comes, in part, from an ability to recognize behaviors as correct or incorrect. AUS that have been programmed to perform in a particular fashion may or may not exhibit behaviors that are recognizably correct while optimal for the given situation. Knowing that such situations can exist may also push an operator to show complacency when observing odd behaviors because they can be explained as possibly correct if not humanlike.

4 The Roles for Learning

4.1 Developing Team Tactics

A common approach to the design of autonomous systems is to design the entire system to be scripted. That is, a human decides on the action the autonomy takes for

any given state the system is in. Such systems face many challenges. The first is the significant investment in human resources in the design because every part of the autonomy must be thought out and scripted. Furthermore, the autonomous capability is dependent upon the incorporated knowledge. Therefore, the investment of human resources necessarily includes subject matter experts in the task the autonomy is addressing. Another difficulty is that scripted autonomy is brittle, e.g. unexpected situations can cause the autonomy to not work. This lack of robustness is due to the expansive state space that exists in the real world. Human designers will not be able to test or even anticipate every situation the autonomous system will be exposed to, resulting in a number of states that will not be addressed by the autonomy or be addressed with limited effectiveness. For example, consider a scripted autonomous system designed to deter piracy that assumes there exists only a single pirate threat at any given time. Once such autonomy encounters a situation where there exists more than one pirate, the script will degrade in effectiveness because the situation was not anticipated. Furthermore, once the autonomous system is introduced, pirates can adapt their tactics to counter the system and render ineffective the specific design of the autonomy. Finally, scripted systems often lack scalability. In particular, the designs will be tied to a particular number of autonomous agents, or a particular autonomous system, meaning each time the number of types of unmanned vehicles change, the autonomy for the system must be redesigned.

In a proof of concept experiment, such a scripted system was compared to a cutting edge multiagent learning method, *Multiagent HyperNEAT*. Multiagent HyperNEAT approaches the problem of multiagent learning by focusing on the geometric relationships among agent policies [13]. The policy geometry is the relationship among policies located at particular positions and the team behavior. Because multiagent HyperNEAT is built upon HyperNEAT, it can exploit the same patterns that HyperNEAT is able, such as regularities. Furthermore, because HyperNEAT can encode repetition with variation, it can encode agent policies that share skills and vary in significant ways. Conventional multiagent learning cannot capture such regularities to enable sharing of skills and variation of policy. For a full description of how multiagent HyperNEAT encodes a team of policies see [13,14]. The main idea is to place a whole set of policies within a team geometry and compute their individual policies as a function of their location within the team.

Moving from simulated domains to real robots will mean that situations may occur where the number of agents varies, such as malfunctions or replacements and therefore ideally team size should be dynamically adjustable. The multiagent HyperNEAT approach allows such scaling because it represents team policies indirectly as a function of team geometry. Thus new agents can be added by simply generating the policy for their assigned team position.

The results of the proof of concept experiment are illustrated in the figure below.

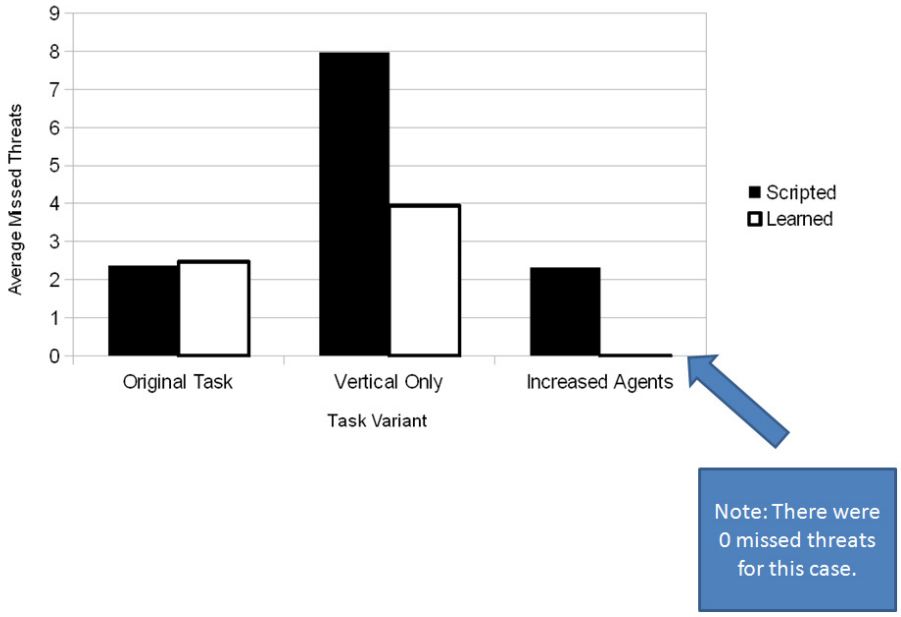

Fig. 1. Performance of Scripted Search versus Learned Policy

Overall, the results show that policies created by multiagent learning approaches are more robust to change. The scripted parallel search and learned multiagent HyperNEAT policies are compared on three variations of a threat detection task. In each of the variations, the policies are tested over 100 evaluations and the results averaged. The learned policy tested is trained solely on the first variation. The first variation is the training task for multiagent HyperNEAT, in which there are seven simulated unmanned vehicles patrolling and the threats can randomly appear along any of the four edges of the operational area. In this task, the learned policy has statistically the same performance as the scripted policy, resulting in the patrols missing 2.37 threats, and the learned patrol policy missing 2.47.

In the second variation, the tactics employed by the threats are altered such that they now appear from two of the four sides at random, thus increasing the density of the attacks along that vector and testing the robustness of the approaches. The learned policy missed 3.94 threats, significantly ($p < 0.001$) outperforming the scripted policy, which decreased significantly in performance to 7.98 threats missed.

In the third variation, threats can appear along all four edges, but the number of simulated unmanned vehicles in the team is increased from seven to eleven. The learned policy exploits the increased number of vehicles, decreasing the missed threats to 0. However, the scripted policy is unable to take advantage of the new vehicles and only insignificantly decreased missed threats to 2.32. These results demonstrate that learning can produce more robust and scalable policies.

4.2 Recognizing Correct Tactical Behavior

Using HyperNEAT to develop team tactics will create more robust and scalable policies and behaviors. However, we must also be concerned with whether or not the human controller will recognize the behaviors as being safe and correct. As the HyperNEAT approach produces Artificial Neural Nets (ANN), we can only look at the team tactics as black boxes, and even within the proof of concept experiment, it took a fair amount of observation of the units to interpret (essentially guess) why they were behaving as they did. A human controller in such a system however, must be able to decide if the tactics being employed are aligned to the mission and whether or not they are properly countering the enemy or handling arising complications in the environment.

One of the primary draw backs to learning behaviors is that in the search for optimal actions the agents can behave in ways that seem foreign and unintelligible to the human operators. It is most likely the case, and something that should be tested, that agents that behave in a more humanlike fashion are more easily trusted by human observers. The development of humanlike agents is possible through hand coding and expert systems, but it is a tedious and complicated process. It is, however possible to learn humanlike behaviors through observation. FALCONET is such a system designed to create high performance humanlike agents through human observation.

Humans learn through several different processes. Learning through observation entails watching the process as performed by some other individual or agent. Learning through experience involves repetitive practice of the process with feedback on performance. Learning can and does occur under each process individually, but it is the combination of observation and experience that generally leads to the highest levels of performance. For instance when learning a new sport humans typically observe others already proficient in the activity before beginning to practice themselves. Observation bootstraps the learning process of experience enabling faster learning speed and higher peak performance.

There is a long history in Machine learning of borrowing from biological systems. Examples include knowledge representations like neural networks, optimization algorithms such as genetic algorithms and ant colony optimization, and learning paradigms like reinforcement learning. In the particular method discussed below the observational-experiential learning cycle is replicated in machine learning to achieve the same goals for simulated learning that are achieved in biological learning.

FALCONET is a method of agent training that follows the biologically inspired cycle of observation and experiential learning. It was designed to enable the creation of high performing, humanlike agents for real time reactionary control systems [15]. Typically, the building of humanlike agents involves the complicated process of interviewing knowledge experts and then codifying that knowledge into a format that is machine readable. This process is complicated and time consuming and has led to the slow adoption of this technique in real world systems, despite the success that can be achieved. This problem is known as the “knowledge engineering bottleneck” [16]. FALCONET was designed to automate the agent creation process from human observation thereby sidestepping the bottleneck.

Previous work has been done using observational data alone to train agents, stopping once an acceptable level of performance is reached on training and validation sets [17,18,19,20,21]. While this might produce humanlike agents, it ignores the possibility that the observational agents will perform poorly in situations not covered in the observational data. The agents when presented with novel situations could perform in unpredictable and unintelligent ways. The experiential phase can fill in these gaps by providing feedback on the agent's performance in novel situations.

The training in FALCONET follows a two phase training approach. First a supervised observational phase, followed by an unsupervised experiential phase. During the observational phase the objective of the learning is to be similar to the actions of a human trainer. Human trainers run through the selected tasks starting from many different scenarios to generate the observational training set. The agents are then trained on this data set while being graded on how closely they mimic the decisions of the human. In the experiential phase the agents are trained further using a measure of performance on the task. In FALCONET all training is done by a hybrid genetic algorithm (GA) particle swarm optimization (PSO) algorithm called PIDGION-alternate. This is an ANN optimization technique that generates efficient ANN controls from simple environmental feedback. FALCONET has been tested showing that it can produce agents that perform as well or better than experiential training alone while incorporating humanlike behaviors. The results from FALCONET also state that unique human operator traits can be incorporated and evident in the final highest performance controls, that is to say that agents sourced from different trainers have slightly different behavioral quirks.

As part of the validation of the FALCONET method experiments were conducted using only the experiential learning phase. High performance controls were created in this manner, but they showed several "improper" quirks, that while more optimal in the performance metric, seem foreign to human operators. These quirks, like driving backward or slamming the controls left and right very quickly, can be programmed out by a human designer, but it requires the a priori knowledge of all "improper" behaviors that would be undesirable. The FALCONET method bypasses this need by bootstrapping the process with human training.

5 Autonomic Control

So far, we have discussed the basic needs that will allow a human commander/controller to exercise command and control over a network of autonomous units that include highly autonomous unmanned systems (AUS). We have recognized that the controller must be able to develop adequate situational awareness of the environment, any enemies, and the behaviors and status of assets available. This SA requires the commander to recognize the behaviors being displayed as aligned with the mission, commander's intent, and applicable procedures. It also requires that these behaviors be robust to the variation found in the environment.

We have also recognized that the human controller is subject to many difficulties inherent to managing automation. Humans are prone to complacency and

automation bias. They also can only work at human speeds and can only handle a finite level of complex information. Abstraction and supervisory control are therefore essential to success if many rapid decisions will need to be made in controlling the network.

We are beginning to model AUS teams utilizing the Rainbow Framework [22] from Carnegie Mellon University. Rainbow groups commands into tactics and strategies and directs the system with those instead of individual actions. This approach allows an automated controller to move the system out of local maximums that it may encounter in utility functions. Additionally, the grouping of actions into tactics and strategies allows for the system to leverage learning techniques and previous human experience in dealing with situations.

Rainbow will provide an autonomic command and control in the sense that it assists with the same set of six tasks, only faster.

Maintain alignment: The mission goals and the commander's intent will be modeled as a set of utility functions within Rainbow. Rainbow evaluates current readings from probes and gauges as well as tactics for changing the resource allocations against these utility functions to select an action.

Provide situational awareness: Rainbow's framework of probes and gauges provides situational awareness into how well the current plan is meeting mission goals.

Advance the plan: The autonomic systems is continuously evaluating the readings from the probes and gauges against the plan and makes changes to adjust in the event that desired goals are not being met.

Comply with procedure: Procedural guidelines can be coded in the tactics employed by Rainbow in the stitch language. It is our intention to also link tactical procedures to learned behaviors.

Counter the enemy: As the environment or enemy actions impinge on success of the goal, Rainbow adjusts the operation based on evaluating tactics against the likelihood of success. The evaluation processes will need to be robust enough to estimate how the changes will effect operations.

Adjust apportionment: The basic types of tactics employed in Rainbow to date have been apportionment decisions. In [22] experiments were done on video teleconferencing services where additional servers were brought online to solve problems that occurred during operations.

In the application of Rainbow, AUS teams are represented in a similar fashion as a network of servers would be. However, we include probes into the physical world providing information both on the AUS and on the environment. Many of these probes relate directly to the sensors that are onboard typical AUS. Strategy decisions involving costs include physical costs of fuel as well as risks found only in systems that interact with the physical world. Likewise, rewards are based on the ability of the AUS to effect a positive change on the environment, often in the form of achieving a probability of detection of other physical entities in a portion of the environment**.**

We have developed an initial proof of concept in autonomic control of unmanned systems by applying the Rainbow Framework to a simulated domain. In this domain, a number of AUS must maximize the probability of detection, P(d), in an environment by maximizing sensor coverage across the area. In this case, P(d) is a

simple metric defined as the fraction of horizontal and vertical paths across the space that do not have sensor coverage across them, i.e. straight paths that can be traversed without detection. Thus each AUS has a location (x,y-coordinate) and sensor range along with other parameters. Because AUS are conceptually similar to computational services (e.g. servers), they can be similarly modeled in the Acme architecture model language that defines a system architecture in the Rainbow Framework. A simple AUS architecture definition is as follows:

```
Component Type AUST extends ArchElementT with {
   Property x : float <<  default : float = 0.0; >> ;
   Property y : float <<  default : float = 0.0; >> ;
   Property fuel : float <<  default : float = 1.0; >> ;
   Property fuelExpendRate : float <<  default : float = 1.0E-4; >> ;
   Property speed : float <<  default : float = 0.01; >> ;
   Property sensorRadius : float <<  default = 0.1; >> ;
   Property cost : float <<  default : float = 1.0; >> ;
}
```

Properties, such as the geographic location and fuel state, represent values that are probed from the (simulated) world. Such values inform the Rainbow model manger, allowing it to accurately reflect and gauge the real system within the Rainbow defined model.

A key feature of Rainbow is the definition of constraints that the system must follow. For example, a web business may desire the minimization of response time for its customers and define a constraint that the response time experienced by any customer if below some threshold. In turn, Rainbow would probe these response time values from the real system and then evaluate the model for constraint violations. If a constraint is violated, Rainbow adapts the model through predefined strategies and then executes these strategies on the real system through effectors. In this proof of concept, the constraint is that the value P(d) must be above a given threshold of 0.8. To satisfy this constraint, Rainbow implements a simple strategy.

```
strategy BruteDetection
[styleApplies && cViolation] {
    t0: (overlapExists) -> move(){
        t0a: (default) -> done;
    }
    t1: (cViolation  && ! overlapExists) -> enlistAUS() {
        t1a: (overlapExists) -> move();
    }
```

In brief, the strategy states that if an overlap in sensor coverage exists, move the active AUS to minimize the overlapping coverage. If there is no overlap, but the constraint is still violated, then add a new AUS to the domain. This strategy implements two tactics that we described in stitch: move and *enlistAUS*. As an example of how these tactics are defined, *enlistAUS* is as follows:

```
tactic enlistAUS () {
      condition {
      // Probability of detection is below a threshold
      ModelAlt.probabilityDetection(M.components) <M.MIN_PDETECT;
      // there should be enough available AUS
      ModelAlt.availableServices(T.AUST) >= 1;
      }
      action {
      set aus = Set.randomSubset(ModelAlt.findServices(T.AUST), 1);
      for (T.AUST freeAUS : aus) {
             S.activateAUS(freeAUS);
             }
      }
      effect {
      // Probability of detection rising should result
      ModelAlt.probabilityDetection(M.components) >= M.PDETECT;
      }
}
```

In the *enlistAUS* tactic, the condition first checks whether the constraint is violated and then if there are any AUS not currently active. If both these conditions are satisfied, a random free AUS is chosen and activated. The effect being that the addition of the new AUS increases the sensor coverage, thus improving P(d) over the current levels.

In the domain, the AUS exist in a two-dimensional plane with coordinate values in the range [0,1]. Initially, no AUS are active, thus the constraint is violated at the start. This compels Rainbow to adapt the system with the above strategies and tactics to achieve the pre-determined desired P(d) level. When each AUS is activated, they are placed at location (0,0) and then move from there. Each AUS moves at the same fixed speed of 0.01, have a sensor range of 0.1, and begin with 100% fuel.

Results demonstrate that Rainbow can be implemented to effectively control such systems. Figure 2 shows that the system begins at a low P(d), indicative of the initial state of the system. However, by time step 150, Rainbow has successfully adapted the system to achieve the desired P(d) value.

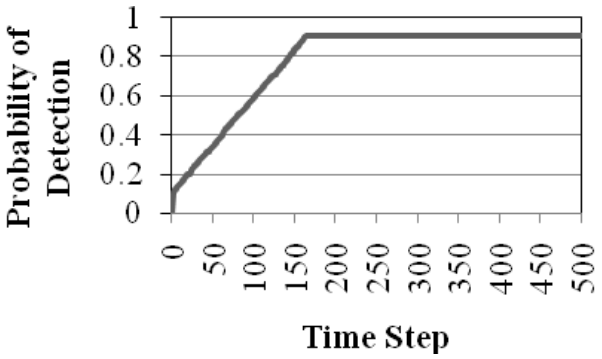

Fig. 2. Probability of Detection in Proof-of-Concept Experiment

Not only is the result interesting, but the behavior of the system is as well. Figure 3 shows the final configuration of the system, when it achieves the P(d) threshold required. Each circle represents an AUS sensor coverage.

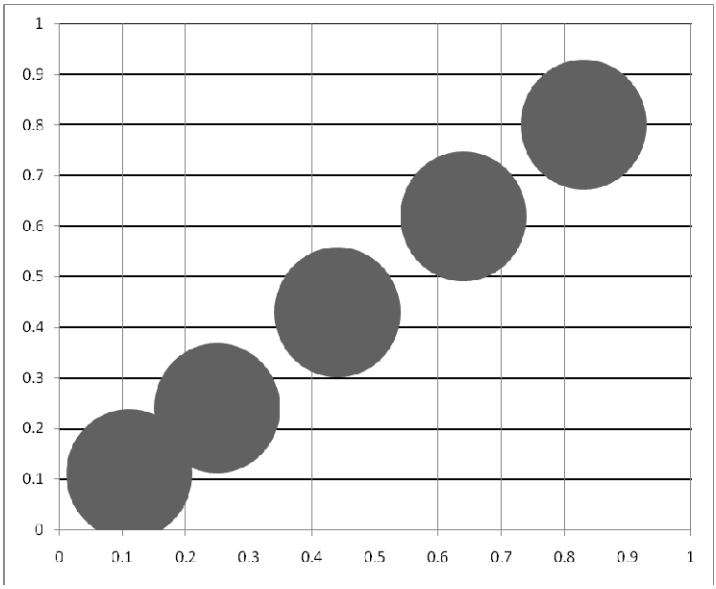

Fig. 3. Final AUS Positions with Sensor Radii

Through the composition of simple strategies and tactics, organization emerges that effectively minimizes the probability of anyone passing through the region undetected. In this simulation, utility of the system is equal to the probability of detection. However, Rainbow includes the capability to calculate system utility as a function of multiple variables. For example, maintaining sensor coverage may be only one important aspect; another may be reducing fuel consumption or minimizing the AUS required. Rainbow weights each of these contributions of utility to determine the overall utility of the system. Through these relative weightings, different aspects can be emphasized. A suite of strategies to address these differing concerns may be required, forming a pareto-front of performance depending on particular user needs.

The ability of Rainbow to automatically and quickly implement strategies frees up the controller to focus on macro level concerns, such as overall probability of detection, fuel levels, and costs, rather than micro-managing individual AUS. Thus the controller can make decisions about required detection levels versus preferred fuel levels and leave it up to Rainbow and its strategies to implement the decisions. Many avenues remain for exploration in the Rainbow Framework including, but not limited to, performance with "human-in-the-loop" changing the system constraints and goals, integrating machine learning into tactics and strategies, extending Rainbow to be able to dynamically acquire system architecture, and evaluating robustness to failures in the system, such as an AUS malfunctioning, being destroyed, running out of fuel, or being reassigned.

6 Conclusions

We have both found and produced, proof-of-concept level experiments that demonstrate possible solutions to some of the challenges we perceive for the successful command and control of teams that include AUS. Our goal is to continue to pursue these possible solutions.

The command and control of teams requires that commanders be able to work at a suitable level of abstraction. Commanders must be able to recognize when changes to a plan are required and must have the ability to affect such a change. The dynamic nature of the military environment indicates the need for robust adaptable capabilities for decision making in the individual AUS, but also the ability for their actions to be recognizable to human controllers. Autonomic capabilities are a likely approach to allow commanders to handle very large teams that may require rapid decision making, but the autonomic strategies must also be made more adaptable and in doing so also maintain the property of being recognizable by a commander.

References

1. Willard, R.F.: Rediscovering the Art of Command & Control. Proceedings of the US Naval Institute (2002)
2. Rodas, M.O., Szatkowski, C.X., Veronda, M.C.: Modeling Operator Cognitive Capacity in Complex C2 Environments. In: 16th International Command and Control Research and Technology Symposium (2011)
3. Parasuraman, R., Wickens, C.D.: Humans: Still vital after all these years of automation. Human Factors 50(3), 511–520 (2008)
4. Parasuraman, R., Molly, R., Singh, I.L.: Performance consequences of automation induced "complacency". The International Journal of Aviation Psychology 3(1), 1–23 (1993)
5. Wiener, E.L.: Cockpit automation. In: Wiener, E.L., Nagel, D.C. (eds.) Human Factors in Aviation, pp. 433–461. Academic, San Diego (1988)
6. Parsasuraman, R., Manzey, D.H.: Complacency and Bias in Human Use of Automation: An Attentional Integration. Human Factors 52, 381–410 (2010)
7. 32nd Army Air and Missile Defense Command: Patriot Missile Defense Operations during Operation Iraqi Freedom, Washington, DC (2003)
8. Chen, T.L., Pritchett, A.R.: Development and evaluation of a cockpit decision-aid for emergency trajectory generation. Journal of Aircraft 38, 935–943 (2001)
9. Johnson, K., Ren, L., Kuchar, J., Oman, C.: Interaction of automation and time pressure in a route replanning task. In: International Conference on Human-Computer Interaction in Aeronautics, pp. 132–137 (2002)
10. Layton, C., Smith, P.J., McCoy, E.: Design of a cooperative problem-solving system for en-route flight planning: An empirical evaluation. Human Factors 36, 94–119 (1994)
11. Mosier, K.L., Skitka, L.J., Dunbar, M., McDonnell, L.: Aircrews and automation bias: The advantages of teamwork? The International Journal of Aviation Psychology 11(1), 1–14 (2001)
12. Leli, D., Filskov, S.: Clinical detection of intellectual deterioration associated with brain damage. Journal of Clinical Psychology 40(6), 1435–1441 (1984)
13. D'Ambrosio, D.B., Stanley, K.O.: Generative encoding for multiagent learning. In: Proceedings of the Genetic and Evolutionary Computation Conference (2008)

14. D'Ambrosio, D.B., Lehman, J., Risi, S., Stanley, K.O.: Evolving policy geometry for scalable multiagent learning. In: Proceedings of the Ninth International Conference on Autonomous Agents and Multiagent Systems (2010)
15. Stein, G.: FALCONET: Force-feedback approach for learning from coaching and observation using natural and experiential training. Ph.D. Thesis, University of Central Florida (2009)
16. Feigenbaum, E.A.: Knowledge Engineering: The Applied Side of Artificial Intelligence. Annals of the New York Academy of Sciences 426(1 Computer Culture: The Scientific, Intellectual, and Social Impact of the Computer), 91–107 (1984)
17. Dejong, G., Mooney, R.: Explanation-based learning: An alternative view. Machine Learning 1(2), 145–176 (1986)
18. Lee, S., Shimoji, S.: Machine acquisition of skills by neural networks. In: IEEE International Joint Conference on Neural Networks, vol. II, pp. 781–788 (1991)
19. Sammut, C., Hurst, S., Kedzier, D., Michie, D.: Learning to fly. In: Proceedings of the Ninth International Workshop on Machine Learning, pp. 385–393 (1992)
20. Henninger, A.E., Gonzalez, A.J., Georgipoulos, M., DeMara, R.F.: The limitations of static performance metrics for dynamic tasks learned through observation. Ann Arbor 1001, 43031 (2001)
21. Fernlund, H.K.G., Gonzalez, A.J., Georgiopoulos, M., DeMara, R.F.: Learning tactical human behavior through observation of human performance. IEEE Systems, Man, and Cybernetics, Part B: Cybernetics 36(1), 128–140 (2006)
22. Garlan, D., Cheng, S., Huang, A., Schmerl, B., Steenkiste, P.: Rainbow: Architecture-Based Self Adaptation with Reusable Infrastructure. Computer 37(10), 46–54 (2004)