Acquiring User Models to Test Automated Assistants

Marc Pickett[1] and **David W. Aha**[2] and **J. Gregory Trafton**[2]
[1]NRC/NRL Postdoctoral Fellow; Washington, DC 20375
[2]Navy Center for Applied Research in Artificial Intelligence;
Naval Research Laboratory (Code 5510); Washington, DC 20375
(marc.pickett.ctr | david.aha | greg.trafton) @nrl.navy.mil

Abstract

A central problem in decision support tasks is operator overload, in which a human operator's performance suffers because he or she is overwhelmed by the cognitive requirements of a task. To alleviate this problem, it would be useful to provide the human operator with an *automated assistant* to share some of the task's cognitive load. However, the development cycle for building an automated assistant is hampered by the testing phase because this involves human user studies, which are costly and time-consuming to conduct. As an alternative to user studies, we propose acquiring *user models*, which can be used as a proxy for human users during middle iterations, thereby significantly shortening the development cycle for rapid development. The primary contribution of this paper is a method for coarsely testing automated assistants by using user models acquired from traces gathered from various individual human operators. We apply this method in a case study in which we evaluate an automated assistant for users operating in a simulation of multiple unmanned aerial vehicles.

1 Introduction

Operators of unmanned vehicles can suffer from information overload, which may lead to loss of situation awareness and sometimes fatal consequences (Shanker and Richtel 2011). To address this, researchers have proposed using decision aids with two primary components: a cognitive model that assesses an operator's state in the context of the system they are using to monitor these vehicles and an automated assistant that aids the operator as needed (Ratwani, McCurry, and Trafton 2010). For example, when the cognitive model predicts that the user is distracted the assistant could draw the user's attention and make recommendations, or even execute recommended actions under limited conditions.

Typically, testing these decision aids involves assessing their ability to assist operators in subject studies. However, these studies require substantial time and effort to design and conduct, and a single study is limited to testing at most only a few decision aid variants. Thus, if many design options of the automated assistant need to be considered, testing each all of them can be prohibitively expensive.

Table 1: **Method for Acquiring and Using User Models**

1. Gather traces from human operators in an initial subject study.
2. Acquire user models from traces.
   - Extract feature vectors from traces.
   - Construct an expert operator.
   - Hobble the expert operator to match user feature vectors.
3. Evaluate the user models.
   - Learn a user classifier.
   - Extract traces from the user models.
   - Attempt to fool the classifier with the models' traces.
   - If the model fails to fool the classifier, goto Step 2.
4. Iterate assistant design using the user models:
   - Build/modify an automated assistant.
   - Use the models to test the assistant.
5. Test the assistant with human operators.

We present an alternative method for this task: Derive a set of user behavior models (from data collected from an initial subject study), use them as surrogates to test different designs for the assistant, and focus subsequent subject studies on the ones that performed well. The potential benefit is a large savings in time and cost.

In this paper, we describe this method for acquiring user models in Section 2 and its initial application in Section 3, where the goal is to assist users with controlling a set of (simulated) unmanned air vehicles (UAVs). Our application involves RESCHU (Boussemart and Cummings 2008) scenarios that are designed to elicit information overload conditions. We describe related work in Section 5, the limitations of our method and for drawing lessons from this single case study in Section 4, and conclude in Section 6 with suggestions for future research.

2 Method

An overview of the method for acquiring user models and using these for developing automated assistants is shown in Table 1. In a traditional development cycle, developers build an automated assistant, then test the performance of the assistant by doing human user studies. Because user studies are costly and time-consuming, this can lead to slow iterations. In our approach, we use automated *user models* instead of actual people for some of these iterations. User models are fully generative in that they are given the same

observations that a human user would be given, and must produce specific actions like a human user would. User studies using these models are fast, essentially free, and require no human subjects approval (IRB). Thus, they provide a fast approximation to humans to quickly ferret out significant problems with the automated assistant. The questions arise of how these models are acquired (Steps 1 and 2), how much these models operate like human users (Step 3), and how the models may be used in lieu of humans (Step 4). Finally, the goal of the user model is not to eliminate human user studies completely, but to reduce reliance on them. Because the user models are only approximations to human users, any final assistant should be tested using a user study with actual humans (Step 5).

Step 1. Gathering Traces from Human Operators.

Step 1 is to gather traces from human operators. A trace is a recording of a user's actions and the observations of the environment given to him or her over the course of a single trial in the decision support task. We collect n trials (typically about 10) from each of m users (8 or 9, but more would be preferable). We assume that these trials are independent. That is, we assume that the user does not significantly alter his or her strategy through learning or fatigue during his or her n trials. In practice, we reduce the amount of in-trial learning by allowing the user ample time to familiarize him or her self with the task before recording their traces. Aside from a unique randomly assigned ID number, we record no identifying information about each user (e.g., gender or age).

Step 2. Acquiring User Models from Traces

Step 2 is acquiring models from the traces collected in Step 1. Given a set of $m \cdot n$ traces (n traces for each of m users), we acquire a model for each user by extracting feature vectors from his or her traces (where a feature vector acts as or sort of summary or footprint for each trace), then build the model so that the feature vectors extracted from the model's traces are similar to those extracted from the user's actual traces. Thus, we can view our set of user models as variations of a single parameterized model that takes as input a set of feature vectors, and interacts with the environment to generate similar feature vectors. The benefit of this is that a variety of user models allows us to test how an automated assistant will perform for different types of users.

The feature vectors we extract should be broad enough to encompass the user's interaction style. For example, if we simple extract the trace's performance score (but no other metric), it might be possible to achieve this score with widely different interaction styles (especially for lower scores). With many interdependent features, it becomes less likely that we can generate a model for a user that has these feature vectors, but operates in a markedly different style from the user.

Once we have extracted our traces and feature vectors from them, we will want to know which of these features are most useful for characterizing our users and distinguishing them from each other. We use a multiclass feature weighting algorithm to help us identify which features are best for distinguishing individual users (in the case study below, we use a variant of RELIEF (Kira and Rendell 1992)). Essentially, we search for features that have high variance between users, but low variance for vectors generated by a single user.

Given a set of weights signifying the relative importance of individual features, we build a model by building an "expert" operator, then hobbling it so that feature vectors extracted from its traces match those generated by a particular human operator. The assumption here is that individual humans would perform perfectly except for their constraints, and these constraints account for much of the variation among individuals. For example, RELIEF might tell us that "reaction time" is an important feature to emulate. If our expert operator's reaction time is negligible, and we read a set of traces for user u whose average reaction time is 310ms (with a standard deviation of 20ms), we restrict our expert operator to respond to an events 95% of the time between 270 and 350ms (i.e., what would be statistically expected given a Gaussian with $\mu = 310ms, \sigma = 20ms$). Note that the expert operator need not perform optimally.

Step 3. Evaluating User Models

Step 3 is evaluating the user models produced by Step 2. We would like some assurance that our user models actually operate in the same style as the specific human users they are meant to emulate. We test whether the user models operate like humans by using a sort of automated "Turing Test". That is, we build a classifier that recognizes individual human users by their traces, then we see whether this recognizer classifies the *model* for user u as u. If the classifier thinks that the model for u really is u, then the model is said to pass the automated Turing Test[1].

Given n feature vectors for each of m users, building a classifier is a straightforward multiclass supervised learning problem. We do leave-one-out cross-validation on a suite of classifiers with different parameters, choosing the classifier and parameters that has the best average performance on the test set.

Given a classifier and a set of feature vectors for user u, we build a model for u as described in Step 2. With this model, we generate t traces and extract feature vectors from these traces. We then send these feature vectors to the classifier and count the number of feature vectors that the classifier classified as being generated by user u.

If our classifier has low accuracy for classifying human users, we should extract more features from the users' traces (resulting in longer feature vectors). If the classifier has high accuracy, but the traces generated by the user model are easily distinguished from the actual human traces, then the user model is said to fail the automated "Turing Test", and the model should be modified accordingly to better mimic the user.

[1] We use the term "Turing Test" in quotes to distinguish it from its more common usage involving human-level AI. Some important differences between our "Turing Test" and the more common variant are that ours (1) uses a computer (not a human) as judge of similarity, (2) compares user models to other user models (not actual users), and, of course, that (3) the observations involve styles of operation, not a chat terminal where the judge is allowed to interact with the user model.

Step 4. Designing An Assistant Using User Models

Step 4 is using the user models (acquired by Step 2 and proven acceptable by Step 3) to iteratively design an automated assistant. We assume that we have built an initial automated assistant, and we would like to test it. Because it has the same inputs and outputs as a human user, a user model's interface with the decision support task is the same as that for a human. Therefore, to test a user model we only need to define its interaction with the automated assistant, then run the assistant on various users with varying user-model/assistant interactions.

Since our initial human user traces were from a system that had no automated assistant, we must guess how the humans might interact with the assistant. We can parameterize this. For example, we might expect that a human user might take the assistant's suggestions p percent of the time, where p is an adjustable parameter. The results of running a variety of users and parameters on an automated assistant can then be analyzed to inform the next iteration of automated assistant.

Step 5. Testing the Assistant with Human Subjects

Step 5 is testing the final automated assistant with real human operators. This step is the same as in a traditional development cycle.

3 Case Study: A Multiple UAV Control Task

To illustrate our method, we acquired user models for a control task involving multiple UAVs, then used them to test an automated assistant for the same environment. In this case study, we investigate whether our method results in a plausible interaction between an automated assistant and our user models. In particular, we hypothesize that (1) increased use of the automated assistant improves the user models' performance, and (2) the increased performance is more pronounced for models of lower-performing users. Furthermore, we hypothesize that (3) our user classifier will outperform a random baseline, and (4) our user model will be able to "pass the automated Turing Test" (fool the user classifier) more often than a random user model.

The RESCHU Simulator

For our case study, we use a variant of a simulator provided by Cummings *et al.* called Research Environment for Supervisory Control of Heterogeneous Unmanned Vehicles (RESCHU) (Nehme 2009), (Boussemart and Cummings 2008), which simulates a task controlling multiple unmanned aerial vehicles (UAVs).

The operator's goal in this simulation is to engage as many targets as possible during a 5-minute trial. A screenshot of the simulation is shown in Figure 1. The simulator includes 5 UAVs, multiple targets (of which exactly 7 are always available), and hazard areas (of which 12 are always active). A user's available actions are to change UAV target assignments (`change target`), insert waypoints (`insert WP`), move waypoints (`move WP`), delete waypoints (`delete WP`), instruct a UAV to engage a target at which it has arrived (`engage target`), and do nothing (`NOP`). If a UAV passes through a hazard area, it accumulates damage. After a certain amount of accumulated damage, a UAV is incapacitated. Each non-incapacitated UAV is assigned to a target, and moves along its flight path (as determined by the waypoints the operator has inserted) at a fixed rate until it arrives at a target, where it waits for the user to engage the target. When a target is engaged, a new target appears at a random unoccupied location, and the engaging UAV is randomly assigned to an unassigned target.

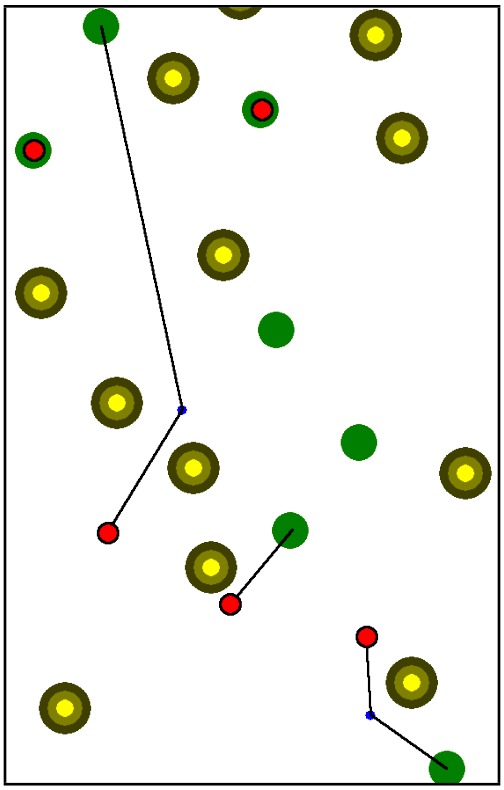

Figure 1: **Screenshot of the supervisory control simulation.** Shown are UAVs (red circles), targets (green circles), hazards (yellow concentric circles), waypoints (blue dots), and projected flight paths (black lines). When a UAV arrives at a target (as the two UAVs have near the top of the screen, each shown as a red circle inside a green circle), it remains motionless until the user instructs the UAV to engage the target, at which point the target is acquired (and a new target appears), and the UAV is randomly assigned to an unassigned target. (See text for details.)

Case Study, Step 1. Gathering Traces

We collected 10 traces from each of 8 human users. Each trace recorded the user's actions and the observations available to him or her while operating RESCHU for a 5-minute period. Though we retained no identifying information about the users, each user is referred to by an index from 1 to 8.

Table 2: **Features and Their Relevance**

Feature	Weight
average distance to hazard before action	.71
tally: `change goal`	.49
bigram tally: `engage target`→`change goal`	.29
bigram tally: `change goal`→`engage target`	.23
tally: `engage target`	.18
bigram tally: `change goal`→`add WP`	.13
bigram tally: `change goal`→`change goal`	.12
tally: `delete WP`	.12
average distance between UAVs	.12
tally: `add WP`	.09
tally: `change WP`	.07
bigram tally: `add WP`→`change goal`	.06
bigram tally: `add WP`→`add WP`	.05
bigram tally: `change goal`→`delete WP`	.05
average UAV idle time	.04
bigram tally: `change WP`→`engage target`	.04
bigram tally: `engage target`→`add WP`	.02
bigram tally: `add WP`→`change WP`	.01
bigram tally: `add WP`→`delete WP`	.01
scores per action	.00
tally: `NOP`	.00
average time between hazard cross and action to fix	.00
(All other bigram tallies)	.00

Case Study, Step 2. Acquiring User Models

Because the number of possible observations a user may receive in RESCHU is large ($\sim 10^{124}$), building an observation-action model, such as that used by (Sammut et al. 1992) and (Šuc, Bratko, and Sammut 2004), was infeasible without a significant amount of state abstraction. Instead, we followed the method described in Step 2 by extracting feature vectors from the 80 traces, and using RELIEF to help judge which features are most distinguishing for our 8 users (i.e., we want features that have high variability between users, but low variability within vectors generated by a single user). We chose RELIEF over other feature weighting algorithms because of its popularity and ease of implementation.

Feature Vector Extraction From each trace, we generated a set of 36 features. These include tallies over the 6 primitive actions, tallies over the 25 "bigrams" for the 5 non-NOP actions (where `A`→ `B` is the number of times `B` was the first non-NOP action after `A`), the average proximity to a hazard a UAV reached before a user rerouted the UAV, the average distance between (or "spread") of the UAVs, the average time a UAV was idle (how long between when the UAV arrived at its target and when the user engaged the UAV), the average number of (non-NOP) actions the user took divided by the user's performance score (as a measure of action efficiency), and the average amount of time a user waited between when a UAV's path crossed a hazard area and taking action to reroute the UAV. Note that the number of `engage target` actions is the same as the performance measure (the number of targets engaged). These features are shown in Table 2 sorted by their RELIEF score.

Construct an Expert Operator Since reaction time is a limiting factor in human user performance for RESCHU, we were able to build a simple rule-based "expert" operator that outperformed our best human operator. The expert operator uses the following rules:

1. **if** there is a UAV waiting at a target, `engage` that target.
2. **else if** the UAV target assignments are "suboptimal" (using a greedy nearest target to UAV calculation), reassign the targets greedily (using `change goals`).
3. **else if** a UAV's flight path crosses a hazard area, reroute the UAV around the hazard (using `add WP`).

This simple operator performs well, averaging a score of 27.3 targets acquired per 5-minute session, compared to an average of 21.7 targets acquired per session for the human users. The highest scoring human user averaged 24.5 targets per session.

Hobbling the Expert Operator This expert operator served as a basis for constructing individual user models. Based on Table 2, we constrained our expert operator to mimic the user's average distance to hazard before an evasive action, the user's average total number of `change goals`, and the user's average UAV idle time. For example, the expert operator might delay engaging a target to better match the user's average UAV idle time. We chose these features based on their RELIEF weightings and their ease of implementation.

Case Study, Step 3. Evaluating the User Model

Since there is a good deal of interdependence among the features in a user's feature vector, we hypothesize that it would be difficult to generate a trace that produces a similar feature vector, but that operates with a markedly different style. We implement the substeps of Step 3 and learn a classifier that distinguishes human users by their feature vectors, extract traces from the user models, then test whether the classifier can distinguish human users from their models.

Learning a User Classifier For each of our 8 human users, we have 10 vectors of 36 real-valued features. Thus, learning a classifier is a straightforward application of multiclass supervised learning, with the classes being the 8 individual users. We trained a suite of 35 classifiers using leave-one-out cross-validation. Included in this suite were various paramaterizations of Bayes Logistic Regression, Naive Bayes, Gaussian Process Classifier, k-Nearest Neighbors, Least-Angle Regression, Sparse Multinomial Logistic Regression, Elastic Networks, and Support Vector Machines (SVMs) with various kernels. The best of these classifiers for our data was an SVM with a polynomial kernel (Poly-SVM), which yields an average of 62.5% accuracy on the test set (compared to 12.5% accuracy for a random classifier).

Fooling the Classifier Using the user model, we generated 10 new traces for each of our 8 users with the aim of mimicking that user. Poly-SVM's classification accuracy was 27.5% on these traces. That is, given a trace generated by the model for user u, the classifier correctly identified the

trace as coming from user u 27.5% of the time. A random classifier gives us 12.5% accuracy. Given our classifier, we consider a practical upper limit to be 62.5% because this is the accuracy on actual human traces.

Given these numbers, the user models are said to have done poorly on the automated "Turing Test". In practice, we would want both the classifier and the model's accuracy to be much higher, and we would iterate more times on Step 2 to improve these figures. As the current work is a demonstration of principle, we hope to improve these figures in future work.

Case Study, Step 4. Testing an Automated Assistant

In this step we describe how we built our automated assistant, and how we used our user models to test it.

We implemented an automated assistant that constantly "suggested" an action —the action the expert operator would do— at every step of the session. The user (or the user model) then had the choice of executing a primitive action, or calling the assistant, which would then execute its suggested action.

We had no model for how users would use an assistant (as there were no assistants available to them when we collected the traces), so we parameterized the user models' interaction with the assistant. In particular, we introduced a parameter p that denotes the percent probability that a user model will call the assistant at any particular time.

We ran the 8 user models varying p from 0 to 100%. Results are shown in Figure 2. Individual user models' performances are shown as light lines, with the average performance shown as bold. From this figure, we see that while the increased use of the automated assistant improves performance for all user models, the user models with the lowest non-assisted performance benefit the most from the model. When $p = 0$, each model performs as if there is no assistant. When p is set to 100%, the models all converge to operating identically to the expert operator. In practice, $p = 100\%$ would be feasible for our case study because the number of non-NOP suggestions given by the automated assistant averages 1 every 3.3 seconds (and the assistant rarely gives more than one non-NOP suggestion per second).

4 Discussion

All four of our hypotheses from the beginning of Section 3 were supported by our case study: (1) increased use of the automated assistant resulted in increased performance of the user models, (2) the performance increase was more dramatic for models of lower-performing users, (3) our user classifier was correct an average of 62.5% on the test set, better than 12.5% for a random classifier, and (4) our user models passed the automated "Turing Test" by fooling the classifier 27.5% of the time, better than 12.5% as would be expected for a random classification. However, we hope that this performance can be improved upon in future work.

Since the expert operator performs so well, we might be tempted to simply let the expert operator take over (thus completely automating the task), but doing so would break the requirement that the human operator must maintain situation awareness of the task.

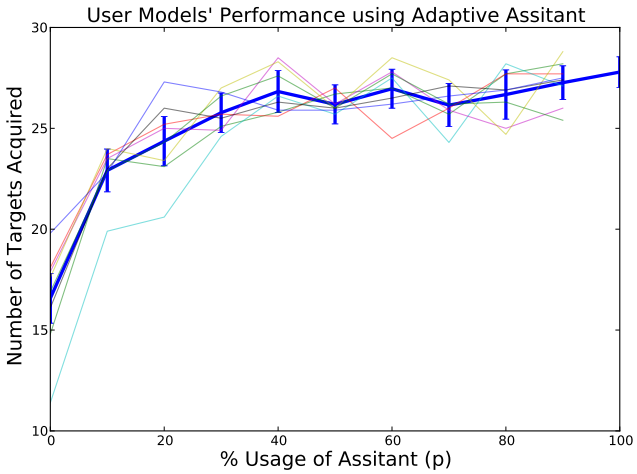

Figure 2: **How the Assistant Helped the User Models.** Individual user models' performances are shown as light lines, with the average performance shown as bold.

5 Related Work

This work differs from most work on Imitation Learning and Cognitive Behavioral Modeling in that it attempts to mimic high-level feature vectors extracted from user traces, rather than the process that generated these traces itself. This includes work using cognitive architectures, such as ACT-R (Anderson 1993), to model user behavior such as driving automobiles (Salvucci 2006) or credibility judgments in micro-blogging (Liao, Pirolli, and Fu 2012). (For a survey of work in Cognitive Behavioral Modeling, see (Zacharias, MacMillan, and Van Hemel 2008).)

Previous work on behavioral cloning characterized the task as a state machine (e.g., as a Markov Decision Process), and mimicked the user's policy (Isaac and Sammut 2003), (Galata, Johnson, and Hogg 1999), (Floyd and Esfandiari 2010), (Argall et al. 2009). As in our case study, the number of possible observations in an environment often makes this approach impractical without state abstraction. Furthermore, much of this prior work focuses on performing well on a specific task, such as flying well (Šuc, Bratko, and Sammut 2004), (Bain and Sammut 1995), instead of attempting to accurately model humans. That is, this prior work focuses on maximizing task performance regardless of whether the behavior is human-like. To our knowledge, no previous work focuses on modeling individual users, as our user models do.

The original developers of RESCHU have learned a user model for RESCHU (Boussemart and Cummings 2008), but this model is descriptive and cannot be used to generate actual operation in RESCHU, unlike our user models. Furthermore, this work does not model individual users, but an aggregate of users.

6 Conclusion

We have introduced a method for rapidly developing automated assistants by acquiring user models, and we demonstrated the method using a case study of RESCHU. Though

we've demonstrated the principle of the method, there are still many issues to be addressed. For example, we have not substantiated the hypothesis that a user model which matches a user's feature vectors also matches the user's style of operation. One future correction for this is an analysis of the interdependence of the user's features, akin to cross-validation. For example, we can leave out the "bigram tally: `engage target→change goal`" feature, then see if a user model that matches the other features also matches this feature. In our case study, we only performed a single iteration of steps 2, 3, and 4. In future work, we intend to extract more features from our user traces (to increase classification accuracy for users), update our models to reflect these new features (to improve classification for the user models), and thereby iterate steps 2, 3, and 4. In our case study, we also left out Step 5, testing the assistant on human users, to validate if the assistant helps out actual humans as well as their models.

Acknowledgements

This research was sponsored by NRL. Marc Pickett performed this work while supported by an NRC postdoctoral fellowship at the Naval Research Laboratory. The views and opinions contained in this paper are those of the authors, and should not be interpreted as representing the official views or policies, either expressed or implied, of NRL or the DoD.

References

Anderson, J. R. 1993. *Rules of the Mind*. Hillsdale, NJ: Lawrence Erlbaum Associates.

Argall, B. D.; Chernova, S.; Veloso, M.; and Browning, B. 2009. A survey of robot learning from demonstration. *Robot. Auton. Syst.* 57:469–483.

Bain, M., and Sammut, C. 1995. A framework for behavioural cloning. In Furukawa, K.; Michie, D.; and Muggleton, S., eds., *Machine Intelligence 15*, 103–129. Oxford University Press.

Boussemart, Y., and Cummings, M. 2008. Behavioral recognition and prediction of an operator supervising multiple heterogeneous unmanned vehicles. In *Humans operating unmanned systems*.

Floyd, M. W., and Esfandiari, B. 2010. Toward a Domain-independent Case-based Reasoning Approach for Imitation: Three Case Studies in Gaming. In *Workshop on Case-Based Reasoning for Computer Games at the 18th International Conference on Case-Based Reasoning, ICCBR 2010, Alessandria, Italy, July 19-22, 2010. Proceedings*, 55–64.

Galata, A.; Johnson, N.; and Hogg, D. 1999. Learning Behaviour Models of Human Activities. In *In Proc. BMVC*, 12–22.

Isaac, A., and Sammut, C. 2003. Goal-directed learning to fly. In Fawcett, T., and Mishra, N., eds., *ICML*, 258–265. AAAI Press.

Kira, K., and Rendell, L. A. 1992. A practical approach to feature selection. In Sleeman and Edwards (1992), 249–256.

Liao, Q.; Pirolli, P.; and Fu, W. 2012. An act-r model of credibility judgment of micro-blogging web pages. *ICCM 2012 Proceedings* 103.

Nehme, C. 2009. *Modeling Human Supervisory Control in Heterogeneous Unmanned Vehicle System*. Ph.D. Dissertation, MIT Dept. of Aeronautics and Astronautics, Cambridge, MA.

Ratwani, R.; McCurry, J.; and Trafton, J. 2010. Single operator, multiple robots: an eye movement based theoretic model of operator situation awareness. In *Proceeding of the 5th ACM/IEEE international conference on Human-robot interaction*, 235–242. ACM.

Salvucci, D. 2006. Modeling driver behavior in a cognitive architecture. *Human Factors: The Journal of the Human Factors and Ergonomics Society* 48(2):362–380.

Sammut, C.; Hurst, S.; Kedzier, D.; and Michie, D. 1992. Learning to fly. In Sleeman and Edwards (1992), 385–393.

Shanker, T., and Richtel, M. 2011. In new military, data overload can be deadly. *New York Times* January 15.

Sleeman, D. H., and Edwards, P., eds. 1992. *Proceedings of the Ninth International Workshop on Machine Learning (ML 1992), Aberdeen, Scotland, UK, July 1-3, 1992*. Morgan Kaufmann.

Šuc, D.; Bratko, I.; and Sammut, C. 2004. Learning to fly simple and robust. *Machine Learning: ECML 2004* 407–418.

Zacharias, G.; MacMillan, J.; and Van Hemel, S. 2008. *Behavioral modeling and simulation: from individuals to societies*. National Academy Press.