NASA Contractor Report 166620

An Application of Adaptive Learning to Malfunction Recovery

Robert Edward Cruz

Contract NAG2-302
May 1986

NASA
National Aeronautics and
Space Administration

NASA Contractor Report 166620

An Application of Adaptive Learning to Malfunction Recovery

Robert Edward Cruz
University of California at Los Angeles, Los Angeles, California 90024

Prepared for
Ames Research Center
Dryden Flight Research Facility
Edwards, California
under Contract NAG2-302

1986

NASA
National Aeronautics and
Space Administration
Ames Research Center
Dryden Flight Research Facility
Edwards, California 93534

TABLE OF CONTENTS

LIST OF FIGURES

page

Introduction

Objectives

As aircraft technology advances in complexity, piloting an aircraft is becoming more difficult and subject to error. This difficulty can be critical during an in-flight malfunction, risking the loss of both the pilot and aircraft. In these situations, it is important to devise automated assistance for the pilot. With this goal in mind, UCLA and the NASA Dryden Flight Research Facility are developing expert systems for potential onboard use in future aircraft. The research presented in this thesis, while a long way from satisfying the goal, represents an initial step towards its achievement.

The immediate objective of this study is to develop a controller that learns an aircraft task and recovers when the aircraft malfunctions. A computer program is used to simulate both the controller and the aircraft. Given limited *a priori* information and a trial-and-error learning strategy, the controller learns to navigate a two-dimensional aircraft through a pre-established mission. The controller uses performance feedback that is taken during and after each aircraft flight. Because its learning strategy is independent of flight dynamics, the model can be applied to both normal and abnormal flight situations.

In essence, the controller decomposes the problem into mutually isolated subproblems corresponding to different regions of the aircraft's allowable state space. For each subproblem, the controller implemen'ts the same problem-solving algorithm. The resulting solutions to each subproblem contribute to the accomplishment of the overall flight task. In this manner, the controller produces useful results for a problem involving a relatively large search space. Additionally, the decomposition technique lends itself to faster computation possibilities related to parallel processing implementations.

Previous Work

The research leading to the present work centers on controllers designed for the cart-pole system shown in Figure 1.

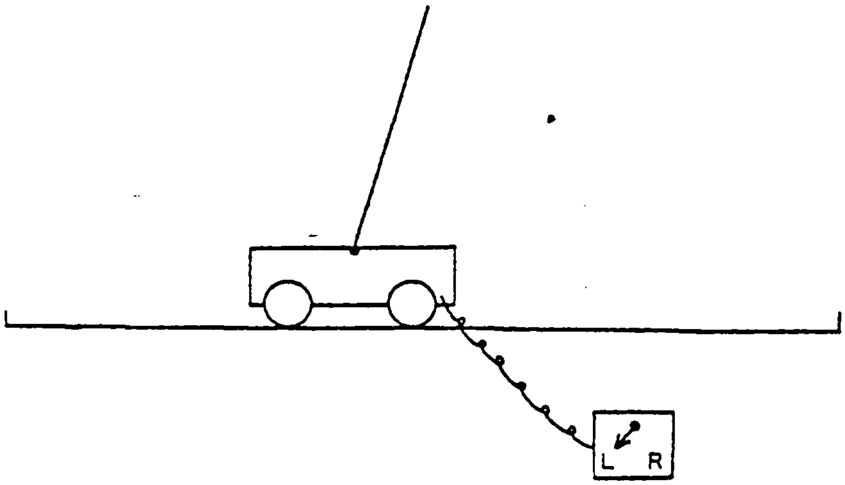


Figure 1. Cart-Pole System

The system consists of a rigid pole mounted to the top of a motorized cart. The cart moves in two directions, left and right, along a straight track of fixed length. The pole is hinged to the cart so that it rotates only in the vertical plane bounding the cart's motion. The controller moves the cart by applying a constant-force motor either to the left or to the right. The cart-pole system is inherently unstable. Therefore, the controller's task is to keep the pole from falling by continually moving the cart left and right as appropriate.

The cart-pole system was initially devised by Donaldson [4] in 1960. In his work, Donaldson designs an automaton that learns the cart-pole balancing task by comparing its control movements to those of a human. This learning strategy, using the terminology of Carbonell, *et. al.* [3] is called "learning by example." The human assumes the role of a teacher who provides examples for the automaton to imitate.

In 1964, Widrow and Smith [13] designed a controller that could be trained to effectively balance the pole. It consists of an encoder and an adaptive linear element, or Adaline. The encoder generates patterns based on the values of four variables that describe the cart-pole system state:

x : the position of the cart on the track,

θ : the angle of the pole with the vertical,

$\dot{x}$: the velocity of the cart, and

$\dot{\theta}$: the angular velocity of the pole.

The encoding scheme partitions each variable into discrete intervals. Consequently, each pattern represents a different combination of intervals occupied by the state values.

The Adaline produces a weighted sum from the encoded patterns. If the sum is greater than or equal to a certain threshold value, the controller applies the cart's motor to the right; otherwise, it applies the motor to the left.

The controller learns to balance the pole by adjusting the Adaline's weights according to an observer's periodic assessment of the controller's performance. When performance improves, it changes the weights to reinforce the Adaline's decision logic. Conversely, when performance degrades, it adjusts the weights so that the decision logic is reversed. When the observer cannot distinguish a change in performance, the weights are left unchanged. Widrow and Smith refer to this learning technique as "selective bootstrapping." Though it does not learn by examples, it still requires a human observer to assess its performance.

In 1968, Michie and Chambers [6], [7] presented an autonomous controller for the cart-pole problem. Its learning strategy, using Carbonell, *et. al.* [3] terminology again, is one of "learning from observation and discovery." Both the controller and the cart-pole system are simulated by a computer program called Boxes. The name derives from the method used to partition the cart-pole state space. In Michie and Chambers' representation, the state variables are plotted along four mutually orthogonal axes. Consequently, each system state corresponds to a unique point in the 4-dimensional state space. By using Widrow and Smith's scheme of partitioning the state variables into intervals, the state space divides into discrete regions, or boxes.

A "demon" resides in each box. Each demon decides the controller's output when the cart-pole state enters its box. By tabulating the consequences of their decisions, the demons learn the best controller output for each cart-pole state. Hence, the controller automatically assesses its performance and adjusts its decisions so that it eventually learns its task.

In 1982, Barto, Sutton, and Anderson [2] presented a similar program for the cart-pole problem. Their aim was to show how the cart-pole controller could be built with neuron-like adaptive elements that they had developed. The controller consists of a single Associative Search Element (ASE) and a single Adaptive Critic Element (ACE). Both elements rely on the state space representation used by Michie and Chambers. The ASE utilizes adaptive threshold logic to control the cart's movement. Its thresholds are modified according to reinforcement feedback provided by the ACE. The ACE produces the feedback by applying threshold logic to the consequences of each controller output. Barto, *et. al.* showed that their controller performs significantly better than the one designed by Michie and Chambers.

In 1966, Schaefer and Cannon [10] showed that the cart-pole problem generalizes to an infinite sequence of problems of increasing difficulty, with 1, 2, 3, etc., poles balanced each on top of the other. The controller to be described represents a different generalization of the problem. Whereas motion in the cart-pole system is one-dimensional, it provides control for a two-dimensional system. Consequently, this research lays the groundwork for future work on automatic control in two- and three-dimensional systems.

Outline of the Paper

The organization of this paper has been divided into three major sections. In the first, the Boxes method is built into a controller for a two-dimensional aircraft model. The controller is exercised in three simulation experiments. In the first experiment, the controller is designed with a learning strategy similar to Michie and Chambers'. Afterward, the controller is enhanced with adaptive elements performing functions similar to the ASE and ACE designed by Barto, *et. al.*. In the second section, two more experiments are conducted. Their purpose is to study the controller's ability to pilot the aircraft after the aircraft malfunctions. The last section is devoted to a discussion of the controller's properties, as well as its performance limitations.

Experiments in Adaptive Control

In the following experiments, a controller is developed for a simplified two-dimensional aircraft model. The aircraft's environment consists of pre-established boundaries on its flight position and velocity with respect to a two-dimensional Cartesian coordinate system (Figure 2).

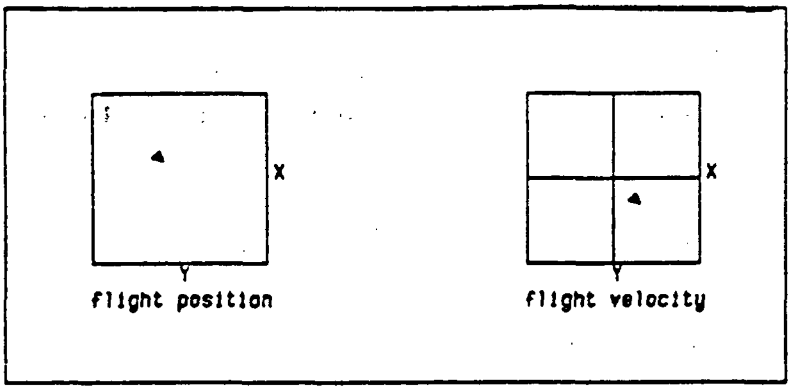


Figure 2. 2-Dimensional Aircraft in a Position-Velocity World

The aircraft is equipped with force actuators that provide constant acceleration in eight directions with respect to the center of its vertical plane of motion: up, down, left, right, up-left, up-right, down-left, and down-right. These actuators correspond to the bi-directional motor used in the cart-pole problem. Therefore, the controller has been designed to activate only one actuator at a time. The aircraft enters a failure state when it flies outside of its position boundaries or exceeds maximum speed limits. These restrictions correspond to the cart running off an end of its track or the pole falling. A flight succeeds when the controller maintains flight within position and velocity limits for a predetermined amount of time.

The controller's design has been adapted from the Boxes system developed by Michie and Chambers [6], [7]. The exact details will be described in the following sections.

Discretization of Aircraft States

At any point in time, the current aircraft state is defined by four variables:

x : the aircraft's position on the X axis,
y : the aircraft's position on the Y axis,
$\dot{x}$: the aircraft's velocity along the X axis, and
$\dot{y}$: the aircraft's velocity along the Y axis.

These variables correspond to the variables x, θ, $\dot{x}$, and θ which defined the cart-pole system state. The variables are plotted along four mutually orthogonal axes. This orientation defines a four-dimensional state space. Each aircraft state is represented by a point in this space. To differentiate between aircraft states, the four state variables are divided into value ranges creating discrete thresholds for the state values. (Figure 3).

The proper threshold values are dependent on performance characteristics of the model aircraft and its mission. For the next three experiments, the thresholds shown in Figure 3 have been selected. The variables x and y are partitioned into five allowable ranges by thresholds at 10, 30, and 50 meters

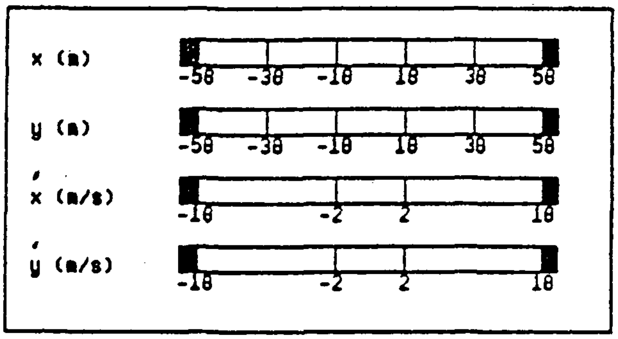


Figure 3. Range-Coded Aircraft State Variables

in both the plus and minus directions with respect to the coordinate origin. Values for x or y of magnitude greater than 50 meters signal an aircraft failure. Similarly, the flight velocity variables $\dot{x}$ and $\dot{y}$ are divided into three distinct ranges by symmetric thresholds at 2 and 10 meters per second. Again, values for $\dot{x}$ and $\dot{y}$ of magnitude greater than 10 m/s constitute an aircraft failure. Thresholding the state variables thus "lumps together" closely-related aircraft states such that the four-dimensional state space within which the aircraft operates becomes subdivided into 5×5×3×3=225 distinct regions, or "boxes." Using the Boxes framework, the controller's task may be regarded as maintaining the four state variables within their limits so that the current aircraft state falls within one of the 225 boxes at all times.

Force Actuator Activation

For simulation purposes, the aircraft's flight has been time-sliced into 1-second intervals. During each interval, the controller activates a force actuator. For the experiments that follow, each actuator has been "designed" to provide 1.5 newtons of constant thrust in one of the eight directions. Depending on the direction in which it is applied, the activation of an actuator changes the aircraft's current position and velocity and, thus determines a new aircraft state. In this fashion, each actuator activation serves as a transitional operator that moves the aircraft from one box to another within its allowable state space.

Problem Decomposition Using Demons

To solve its problem, the controller must learn to avoid (sequences of) actions that lead to an aircraft failure. Obviously, certain actions are appropriate in some instances and inappropriate in others. Because the controller does not have a built-in model of its environment, it must learn by trial and error the proper actuator(s) to activate in a given situation.

Recall that partitioning the state variables has created a four-dimensional state space with 225 regions, or boxes. For illustrative purposes, imagine that these boxes are inhabited by "local demons"--one per box--all of which are under the supervision of a "global demon" (Figure 4). The global demon is in charge of the overall flight task. The local demons concern themselves only with aircraft flight when the aircraft state enters their box. Upon entry into a box, the local demon must decide which of the aircraft's eight actuators to activate next. After making its decision, the local demon informs the global demon who, in turn, activates the apprpriate actuator. After the actuator has been activated for a unit time-step, the global demon determines the new box within which the aircraft state now resides, and asks the corresponding local demon which force to activate next. This sequence continues until the

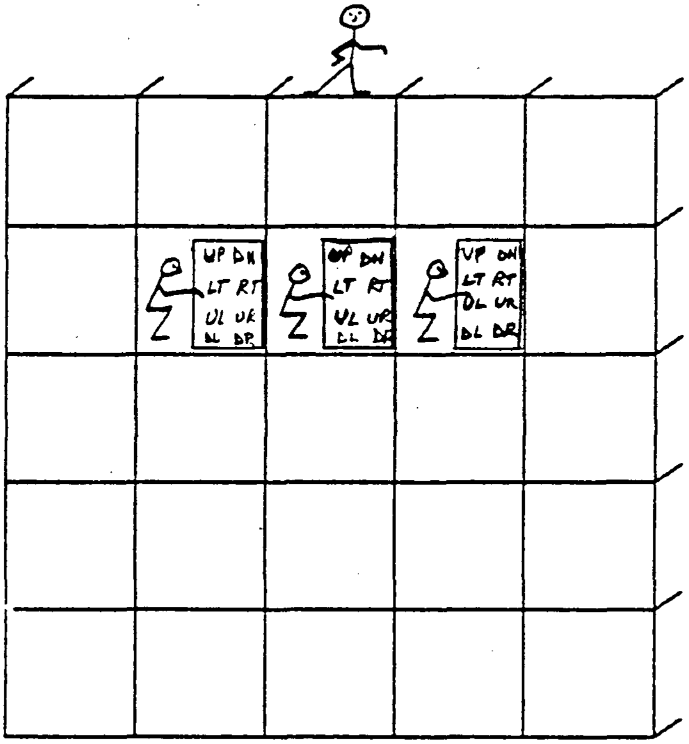


Figure 4. Network of Demons for the Aircraft State Space

aircraft enters a failure state, thus ending the trial run.

The use of global and local demons exemplifies the problem-solving technique of problem decomposition into subproblems. In order to solve the overall problem, the global demon divides it equally into 225 smaller ones and delegates their solutions to the local demons. Because each demon oversees a separate region of the aircraft state space, its job is to determine which force setting best avoids aircraft failure when the current state falls within its assigned region.

In order to carry out its task, each local demon records its previous experience of the aircraft's flight by tabulating the following data:

Force Lifetimes: The total lifetime of decisions to activate a force actuator in a given direction. A force lifetime is the difference between the time of aircraft failure and the time when the aircraft state enters a box and the local demon decides to apply the force. A force's total Lifetime is a weighted sum of all of its "individual" lives during previous runs.

Force Usages: Weighted sums, for each force direction, of the total number of times the local demon decided to activate a force during previous runs.

Entry Times: The times during which the aircraft state entered the demon's box during the current run. Time is initialized to 1 at the beginning of a run, and continues in 1-second increments until aircraft failure.

Experimental Procedures and Results

Three experiments of 1000 simulated flights are conducted. Before the first run of each experiment, local force Lifetimes and Usages are initialized to zero. Additionally, control decisions for the local demons are determined at random. Each run begins at a randomly-generated initial point within the aircraft's allowable state space. The run terminates when the aircraft enters a failure state or avoids failure for 1200 time-steps. Thus, 1200 seconds, or 20 simulated minutes, is established as the duration of a successful flight.

The objective of the first experiment is to demonstrate that Michie and Chambers' Boxes method can be effectively utilized by a controller for a simple aircraft. This objective assumes a close correspondance between the cart-pole problem and the current aircraft task. Hence, the procedures used in Experiment 1 are similar to those outlined by Michie and Chambers [6].

Experiment 1 Procedures

In this experiment, local demons are allowed to decide on only one force actuator to activate per run. Therefore, regardless of how many times the aircraft state enters a demon's box during a run, the demon's control decision remains the same. Initial states for each run consist of randomly-generated values for x and y between ±30m and values for $\dot{x}$ and $\dot{y}$ between ±2m/s. This initialization procedure restricts the initial aircraft state to nine local demon boxes located in the center portion of the aircraft state space.

When the aircraft state enters a demon box during the first run, the following actions occur:

1. The local demon records the time of entry.

2. The local demon signals the global demon to activate a force actuator. The local demon's decision depends on tabulated experience of the consequences of its previous decisions. However, during the first run, this decision is generated at random.

As these actions continue, the aircraft state transitions from one demon box to another until it finally reaches a failure state. This event terminates a trial run and triggers the following actions:

1. The global demon informs the local demons that an aircraft failure has occurred.

2. Each local demon updates its eight pairs of force Lifetime and Usage totals. Based on these new totals, it determines which force actuator to activate (via the global demon) for the duration of the next trial run.

If a force actuator was active before the aircraft failed, its Lifetime and Usage values are calculated as follows:

$$\text{Lifetime} = \text{Lifetime}' \times \text{DK} + \sum_{i=1}^{N} t_f - t_i$$

where N = the number of times that the aircraft state entered the demon box during the run that just failed, and

t_f and t_i correspond, respectively, to the time of aircraft failure, and the individual times of entry into the demon box.

$$\text{usage} = \text{Usage}' \times \text{DK} + \text{N}$$

where DK = 0.99 is a constant multiplier less than unity that weights recent experience relative to earlier experience.

If a force actuator was inactive before the aircraft failure, its Lifetime and Usage values are reduced, respectively, according to the following equations:

$$\text{Lifetime} = \text{Lifetime}' \times \text{DK}, \text{ and}$$

$$\text{Usage} = \text{Usage}' \times \text{DK}$$

In order to determine which actuator to activate next, the local demons refer to a "target" value supplied by the global demon. This value represents the mean lifetime of the aircraft for all previous runs. It is calculated from the global demon's Lifetime (GL) and Usage (GU) values in the following manner:

$$\text{GL} = \text{GL}' \times \text{DK} + t_f$$

$$\text{GU} = \text{GU}' \times \text{DK} + 1$$

$$\text{target} = \frac{\text{GL}}{\text{GU}}$$

Using the global target value, the local demons assess the relative effectiveness, RE, of each of their eight force actuators. RE is calculated as follows:

$$\text{RE} = \frac{\text{Lifetime} + \text{K} \times \text{target}}{\text{Usage} + \text{K}}$$

where K = 20 is a multiplier weighting global relative to local experience.

Incorporating K and the target into the assessment of a local force actuator serves to base the actuator's value on two levels of experience: global experience from the aircraft's mean lifetime over K runs; and local experience from the actuator's Lifetime and Usage totals.

Once the demon has calculated the relative value of each of its force actuators, it chooses the actuator with the highest value as the one to activate during the next trial run (see footnote).

Experiment 1 Results

Because a pseudo-random number generator was used to generate initial aircraft states and decide the local demons' initial control decisions, Experiment 1 was conducted ten times, each time with a different initial seed value. The average results for the ten tests are plotted in Figure 5. The plot shows the average target value versus simulation run number measured after every 50 runs. Notice the direct relationship between the controller's flight experience and the aircraft's mean lifetime. An important statistic not portrayed is the number of successful flights per experiment. On the average, 41 flights out of a thousand were successful.

The results of Experiment 1 demonstrate that the Boxes method may be used for the control of a simplified model aircraft. However, as evidenced by its low success rate, the controller's effectiveness is limited. Because local de-

Because of the way the experiment is initialized, strict adherence to this decision rule results in the local demon choosing its initial force actuator time after time. Therefore, the rule is followed only after a warm-up period during which each force actuator is randomly selected, or sampled, at least once.

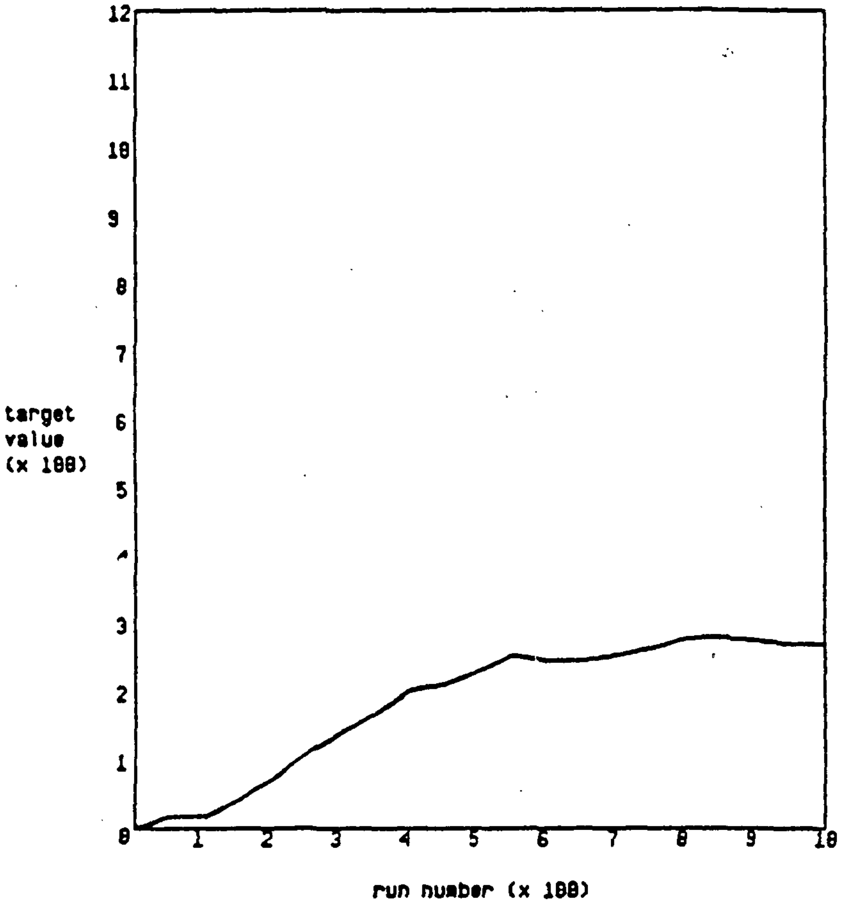


Figure 5. Simulation Results for Experiment 1

cision rules are updated only after each aircraft failure, the demons do not receive feedback as to the immediate consequences of their decisions. Furthermore, restricting the local demons to one force actuator activation per run reduces the controller's flexibility.

The controller's performance can improve by removing these restrictions. The approach taken here will be described in the Experiment 2. It entails making design modifications to the present aircraft controller. The modifications involve the addition of two adaptive threshold-logic elements similar in function to those proposed by Barto *et. al.* [2]. Consequently, the objective of the next experiment is to improve the controller's flight performance.

Experiment 2 Procedures

This experiment differs from the first one in three respects:

1. An Adaptive Critic Element, or ACE, is incorporated into the controller.

2. An Associative Search Element, or ASE, is incorporated into the controller.

3. Local demons may activate more than one force actuator per run.

Otherwise, the initialization procedures, discretization of aircraft states, and local control rules remain the same as those in Experiment 1.

The purpose of the ACE and ASE is to facilitate local learning by constant reinforcement feedback. Recall that, in Experiment 1, local demons had to wait for a failure signal and target value from the global demon before they could update their force actuator values and make a new control decision. With the modified controller, the ASE updates force actuator values every time the aircraft state changes.

Essentially, the function of the ACE is to compare the demon box occupied by the current aircraft state with the box occupied by the previous one, and report its findings to the ASE after each unit time-step. Demon-box comparisons are based on the Lifetime totals for the force actuators activated by the "current" local demon and the "previous" one. The findings, $\hat{r}$, assume the values of either plus or minus one. If the currently activated actuator has a Lifetime as good as, or greater, than that of the previous one, $\hat{r}$ is positive; if not, $\hat{r}$ is negative.

The function of the ASE is to modify local demon force Lifetimes in light of the findings supplied by the ACE. Modification of a demon Lifetime assumes two forms: reinforcement and penalization. Reinforcement occurs when $\hat{r}$ is positive, while penalization corresponds to $\hat{r}$ being negative. Because of the manner in which the ACE calculates $\hat{r}$, good local demon decisions will be reinforced, while poor decisions will be penalized. Note that only demons whose boxes have been entered during the current run become modified; furthermore, modification only applies to the Lifetime for the demon's currently activated actuator.

After each unit time-step, a local force Lifetime is modified according to the following equation:

$$\text{Lifetime} = \text{Lifetime}' + \hat{r} \times \alpha \times \text{e} \times \text{Lifetime}'$$

where $\alpha = 0.05 =$ the minimum percentage of a local Lifetime that may be reinforced/penalized, and

e = an eligibility trace for local demon modification.

The eligibility trace measures the influence of a local demon's actions on reaching the current aircraft state. Obviously, the actions of recently-entered demons have more of an influence than those of distantly-entered ones. Thus, the former demons will have a higher eligibility trace than the latter ones. Eligibility begins at 100% when a demon box is first entered, and decreases exponentially in the following manner:

$$\text{e} = \text{e}' \times \beta$$

where $\beta = 0.95 =$ the percentage of a demon's influence which remains after each simulation time-step.

Experiment 2 Results

As with the first experiment, Experiment 2 was conducted ten times, each time with a different initial seed value. As depicted in Figure 6, the aircraft's mean lifetime was greatly improved by the addition of the ACE and ASE to the controller. In fact, successful flights occurred 537 times out of 1000, or for 53.7% of the trials. In several of the individual simulations, the mean system lifetime approached the upper time limit of 1200 unit time-steps.

Because the controller's task remained the same from Experiment 1 to Experiment 2, the results of the latter experiment may be attributed to the modifications made to the controller. The controller can now make a different decision each time the aircraft state enters a local demon box during the same run. This capability enables the controller to recover more quickly from poor decisions. Additionally, the controller can receive immediate feedback concerning the consequences of local demon decisions. This feedback helps the controller to correlate aircraft performance to local demons' actions.

The results of Experiment 2 show that the modified controller works well at the task to which it was originally assigned. What happens, though, when the controller is assigned a more difficult task? In the next experiment, the controller's task is made more difficult. The ensuing results should provide an idea of the relative tolerance of the controller to changes in task difficulty.

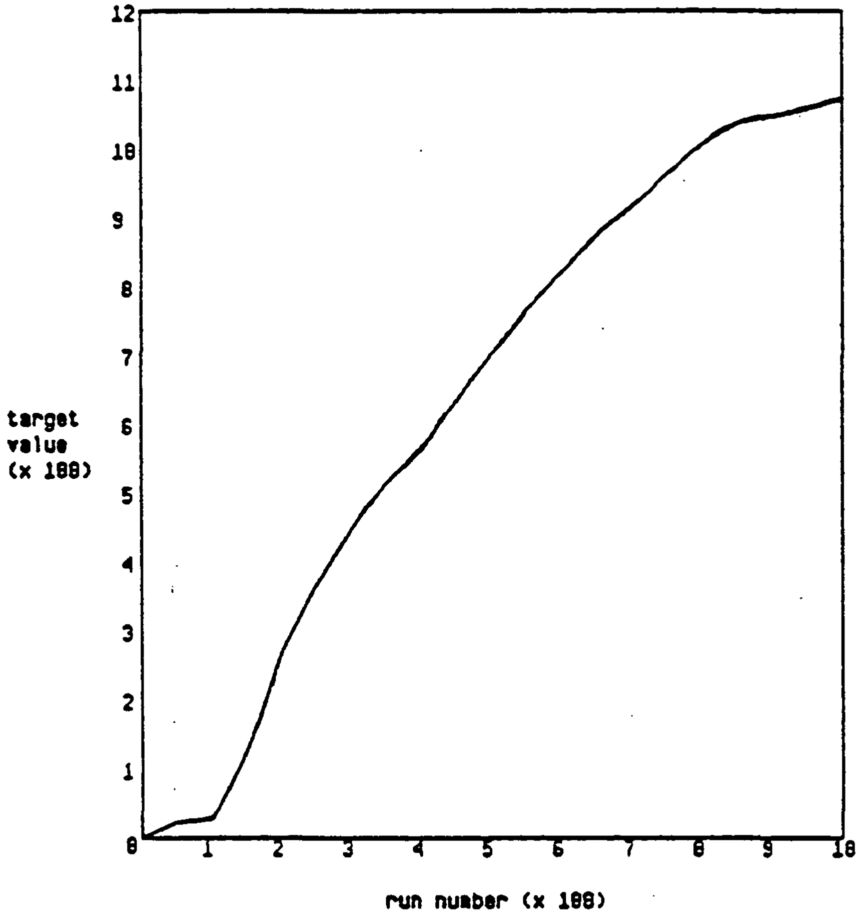


Figure 6. Simulation Results for Experiment 2

Experiment 3 Procedures

This experiment studies the effect on aircraft performance of starting each run from anywhere in the aircraft state space. Therefore, x and y values are randomly selected between ±50m, while the $\dot{x}$ and $\dot{y}$ values are selected from the ±10m/s range. In experiments 1 and 2, initial states fell within only nine possible demon boxes corresponding to the central portion of the state space. In this experiment, all 225 boxes become eligible starting points for a trial run. The change in initialization procedures increases problem difficulty by forcing the controller to map control actions to the entire aircraft state space. Other than this difference, all operating procedures are the same as those used in Experiment 2.

Experiment 3 Results

Experiment 3 was conducted with the same initial seed values used in Experiments 1 and 2. The average results are shown in Figure 7. Notice that aircraft performance is reduced by the addition of 216 more initial states. Also, the average success rate fell to 11.7%. These results show that the controller learns quicker when its starting conditions are more consistent. Otherwise, to attain the same performance reached in Experiment 2, a longer learning period, i.e. more trial runs, are required. This conjecture was not tested.

Comparison of Experimental Results

For comparison purposes, the average results of all three experiments have been superimposed onto the same graph in Figure 8. With respect to the learning curve for Experiment 1, aircraft performance levels out after the first 500 runs. Consequently, the experience gained from the last 500 runs is not utilized. The primary reason for this inefficiency concerns the (long) time intervals between modifications to local demon decision rules. In Experiment 2, local control rules are modified after every unit time-step. The effects on aircraft performance are evident upon inspection of the experimental results. However, in Experiment 3, aircraft performance degrades. This result is a na-

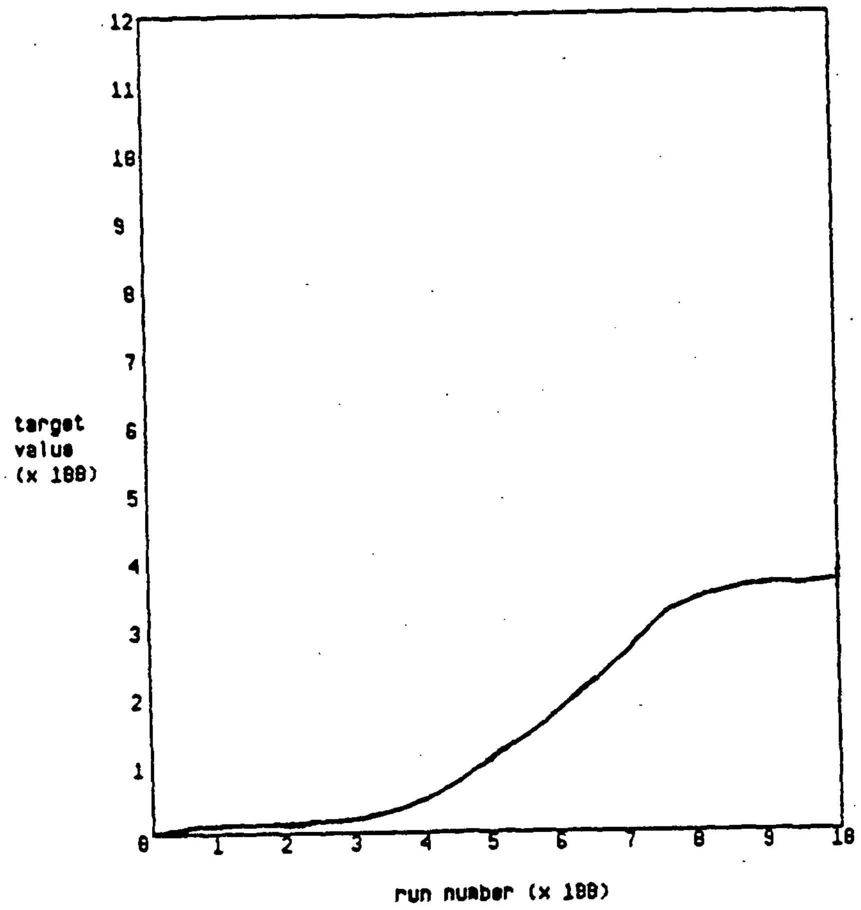


Figure 7. Simulation Results for Experiment 3

tural consequence of the addition of 216 more starting states.

Summary

A controller has been developed for the adaptive control of a simplified model aircraft. Its components include:

1. A global demon that monitors the aircraft state, issues appropriate messages, and activates the aircraft force actuators.

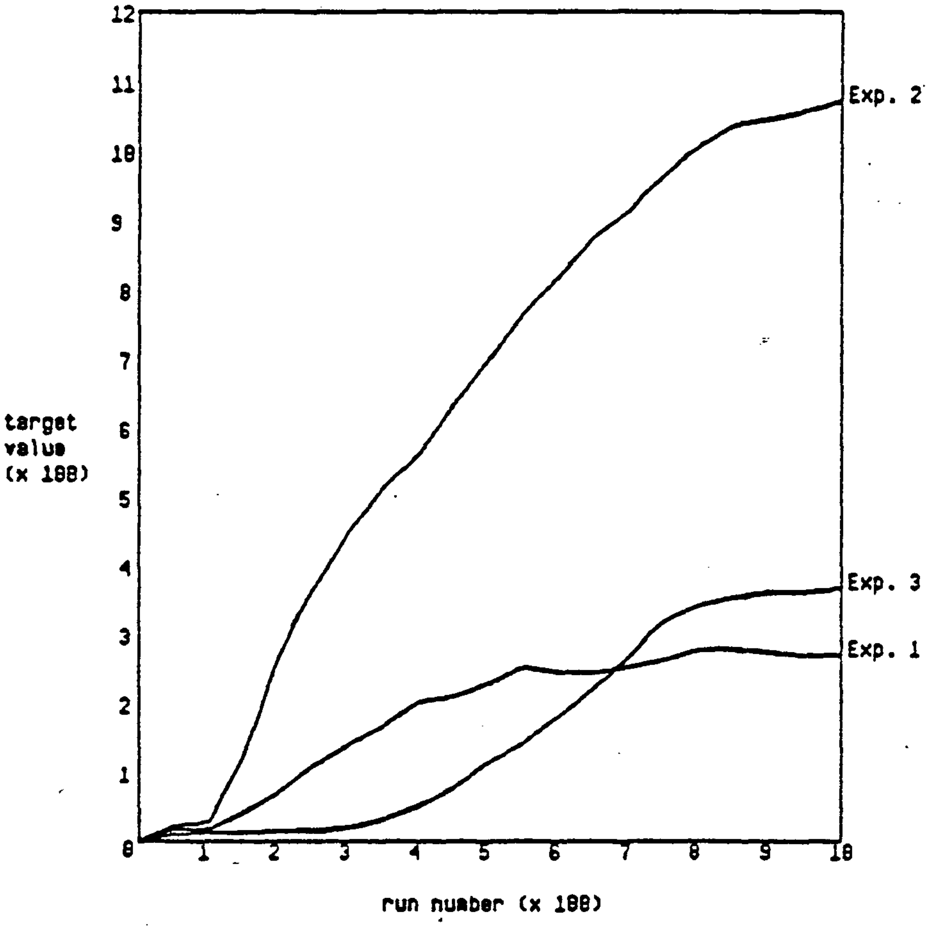


Figure 8. Simulation Results for Experiments 1-3

2. A network of local demons corresponding to different regions of the aircraft state space that advise the global demon of the appropriate actuator to activate when the aircraft state enters a given box. The local demons tabulate data relating to the consequences of their previous control decisions. This data is used to make future control decisions that are implemented by the global demon.

3. Two adaptive threshold-logic elements, the ACE and ASE, that modify local demon control rules in light of immediate aircraft feedback.

Because the controller learns its task from trial-and-error experience, changes in the structure of its components can be made to provide for the control of a more specialized flight task. Such an undertaking is described in the sequel.

Experiments in Malfunction Recovery

The purpose of the preceding experiments was to adapt the Boxes system to a simple flight controller. In achieving this purpose, the experiments provide background for the experiments that follow. Their purpose is to apply the controller to a specific navigational problem, and study its performance under a simulated aircraft malfunction. In the current context, a malfunction exists when the aircraft loses operational control of one or more of its eight force actuators. An important assumption is that, despite the malfunction, the aircraft maintains sufficient directional control to accomplish its predefined mission.

Experiment 4 Problem Description

This experiment proceeds in two phases. In Phase One, the controller learns to pilot the aircraft from one demon box to another. When it achieves proficiency at this task, Phase One ends and Phase Two begins. At this point, an aircraft malfunction is simulated by removing two of the aircraft's eight force actuators. During the second phase, the controller learns to accomplish its original task despite the loss of the actuators.

At the beginning of each run, the aircraft's position and velocity vectors are initialized as follows:

x : -25m

y : 25m

$\dot{x}$: 0m/s

$\dot{y}$: 0m/s

Using the above values for the aircraft state variables, the aircraft's initial configuration is represented in Figure 9.

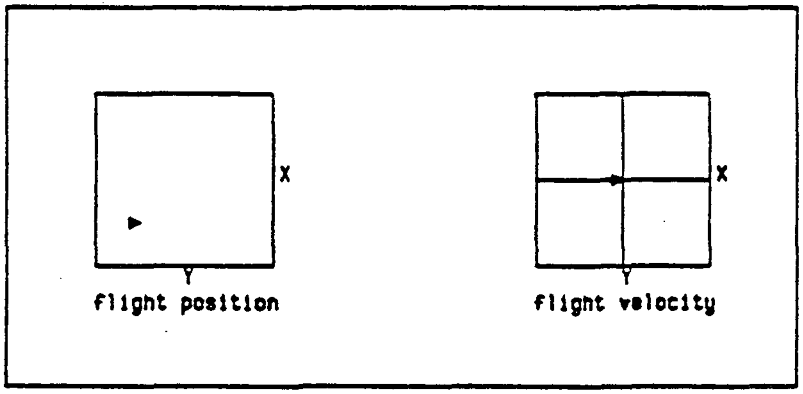


Figure 9. Initial Aircraft State for Experiment 4

From this initial state, the controller must learn to pilot the aircraft to the center box of the disretized state space. This box corresponds to x and y falling within the 0 ±10m range, and $\dot{x}$ and $\dot{y}$ having values between 0 ±2m/s. With respect to the left half of Figure 9, the aircraft must fly from an initial position in the lower-left region of its "airspace" to the center region. As in the preceeding experiments, an aircraft failure occurs when the aircraft exceeds its position and velocity boundaries. Thus, a trial run ends when the aircraft reaches either the goal state or a failure state. Trials continue until the aircraft reaches the goal state 90% of the time. At this point, the aircraft loses operational of two of its eight force actuators. The controller must then recover from this malfunction by learning to complete the aircraft's mission using only six actuators.

Experiment 4 Procedures

Recall that in the preceeding experiments, the aircraft's mission was to prolong flight. Now, its mission is to fly from one demon box to another. Because the mission has changed, the local demons' mutual goal of maximizing their expected lifetimes no longer applies. Instead, the local demons must minimize the aircraft's expected "distance" to the goal. To fulfill this task, local demons tabulate the following data:

Force Distances: Relative approximations, for each force actuator, of the aircraft's distance to the coordinate origin.

Force Usages: Sums, for each force actuator, of the number of times that the local demon decided to activate each actuator during previous trial runs.

To increase the granularity of the state space, thresholds have been added at ±6m/s for the aircraft state variables $\dot{x}$ and $\dot{y}$. The resultant value ranges for the discretized aircraft state space are shown in Figure 10. Consequently, the aircraft state space divides into $5 \times 5 \times 5 \times 5 = 625$ demon boxes instead of the previous 225.

The preceeding discussion outlined two necessary design modifications to the controller. First, the information processed by the local demons has changed. Second, the number of local demons has increased. Now, the sequence of events occurring in a trial run will be explained.

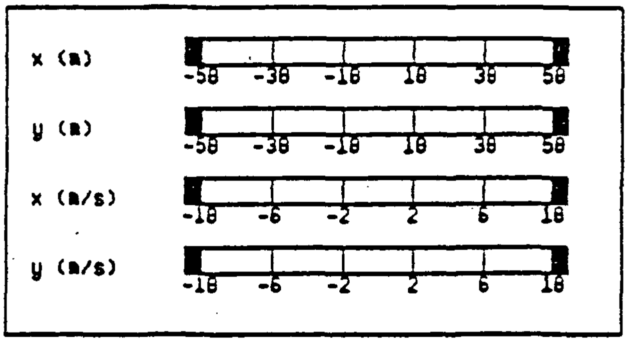


Figure 10. Range-Coded State Variables for Experiments 4 and 5

During each unit time-step, the following actions occur:

1. The global demon signals the local demon whose box has just been entered by the current aircraft state.

2. If the box has never been entered during a trial run, the demon's eight force Distances are initialized with the Pythagorean distance between the current aircraft position and the coordinate origin.

3. The local demon decides on a force actuator for the global demon to activate. The demon makes this decision at random until each of the force actuators has been sampled at least once. Afterward, the demon decides on the force actuator with the lowest Distance:Usage ratio. When the demon has made its decision, it increments its appropriate Force Usage entry by one, and informs the global demon of its decision.

4. The global demon activates the appropriate actuator, which causes the aircraft state variables to change.

5. The ACE compares the current aircraft state with the previous one and reports its findings, $\hat{r}$, to the ASE. To make the comparison, the ACE calculates the Pythagorean distance between the current aircraft position to the coordinate origin. If the current distance is less than the previous one, it sets $\hat{r}$ to 1; otherwise, it sets $\hat{r}$ to -1.

6. The ASE modifies the appropriate force Distance value for each local demon whose box has been entered during the current run. It modifies Distance values as follows:

$$\text{Distance} = \text{Distance}' + \alpha \times \hat{r} \times e \times \text{Distance}'$$

where $\alpha = 0.1$ and

$$e = e' \times \beta$$

where $\beta = 0.8$ (see footnote).

After the ASE modifies the local force Distance entries, the simulated time is incremented a unit step, and steps 1-6 are repeated. This cycle continues until the aircraft reaches either a success or a failure state. Upon success, the ACE issues an $\hat{r}$ value of 1. With $\hat{r}$, the ASE modifies eligible force Distance values as in step 6 above, except that it uses 0.5 as its value for alpha. Upon failure, $\hat{r} = 1$, and the ASE modifies local force Distances using an α value of 3. Consequently, local control decisions are either significantly reinforced (decreased) or penalized (increased) to reflect the end result of the trial run. Afterward, the controller re-initializes the aircraft state variables, and a new trial run begins. The experiment proceeds in 50-run increments. When at least 45 out of 50 flights are successful, Phase One ends.

Phase Two begins with the aircraft losing control of its up-right and down-left force actuators. Despite this malfunction, the controller must regain its 90% proficiency rate for the original aircraft mission. Its control decisions for the six remaining actuators are influenced by the local force Distance and Usage totals gained from Phase One.

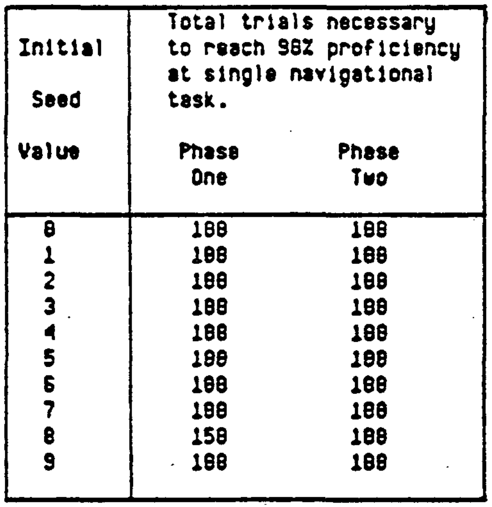
Because of the selection of initial and goal aircraft states, the malfunction prevents the aircraft from flying directly toward its positional goal. Instead, it must combine its up and right actuators to compensate for the loss of the up-right one. Similarly, it must combine the down and left actuators to compensate for the loss of the down-left one. When the aircraft again flies suc-

For a further explanation on the meaning of α, $\hat{r}$, and e, refer to the preceding subsection entitled "Experiment 2 Procedures."

cessful missions 45 times out of 50, Phase Two and the experiment end.

Experiment 4 Results

Experiment 4 was conducted ten times, each time with a different initial seed value. The final results are listed in Figure 11. Notice first that, regardless of the initial seed value, each test achieves the 90% task proficiency rate in both phases of the experiment. This result demonstrates the controller's capability to learn a navigational task under both normal and malfunction conditions. However, the required learning time for each phase does not vary. This result was not expected.



Initial Seed Value	Total trials necessary to reach 90% proficiency at single navigational task.	
	Phase One	Phase Two
0	100	100
1	100	100
2	100	100
3	100	100
4	100	100
5	100	100
6	100	100
7	100	100
8	150	100
9	100	100

Figure 11. Simulation Results for Experiment 4

Initially, Phase One was expected to take longer to complete than Phase Two. Whereas the controller begins Phase One with no experience of the consequences of its decisions, it begins Phase Two with the Distance and Usage totals gained from the previous phase. Therefore, its initial decisions in

Phase Two should be more accurate than the random decisions made at the beginning of Phase One. This initial accuracy was expected to reduce the required learning time in Phase Two.

Two factors contributed to the experimental results. First, the experiment was divided into 50-run increments. At the end of each increment, the number of successful missions was evaluated. If there were at least 45, the appropriate phase would terminate. In these terms, the average time required to complete each phase was two. Perhaps, given a more difficult task (i.e. one that takes longer to complete), the flight experience gained in Phase One would have been reflected in a shorter learning time for Phase Two.

The second factor concerns the particular actuators that malfunctioned. During Phase One, the controller learned that the up-right actuator moved the aircraft closest to the goal from its initial position. However, this actuator was inoperational during Phase Two. Consequently, the controller's "best choice" in the first phase was no longer an alternative in the second. This condition prevented the controller from effectively utilizing its experience gained in Phase One.

To test the validity of these ideas, Experiment 5 was devised. Its aim is to study the effects of task difficulty and actuator malfunction on required learning time. Thus, the final results should show more clearly the temporal relationship between Phases One and Two.

Experiment 5 Procedures

In this experiment, the controller again learns to pilot the aircraft to the center box of the state space. However, its initial position and velocity no longer remain constant. At the beginning of each trial run, values for the state variables x and y are randomly generated in the [-30,-10] and [10,30] ranges while $\dot{x}$ and $\dot{y}$ values are generated in the [-2,2] range. Consequently, the initial aircraft state falls within one of four local demon boxes surrounding the central region of the aircraft state space.

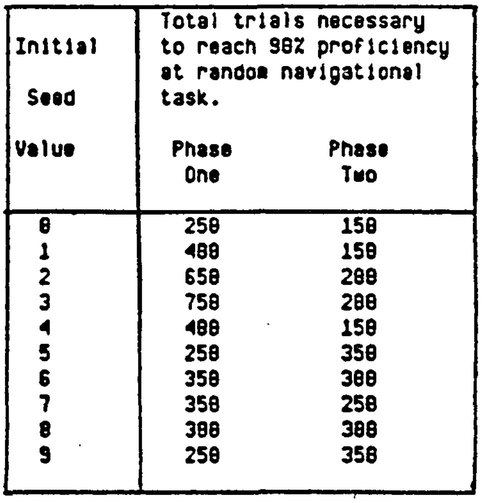
The random initialization procedures are designed for two purposes: (1) to increase the difficulty of the controller's task, and; (2) to increase the accuracy of the controller's initial Phase Two decisions. To clarify this last point, realize that only the up-right and down-left actuators malfunction. Thus, in Phase Two, whenever the aircraft begins in the upper-left and lower-right regions of its airspace, its best decision alternatives--down-right and up-left, respectively--still remain. Thus, half of the controller's Phase Two decisions maximize Phase One experience.

Other than the addition of random initial states, the experimental procedures remain the same as those employed in Experiment 4.

Experiment 5 Results

As usual, Experiment 5 was conducted ten times. The results are shown in Figure 12.

Due to the random initialization procedures, these results vary more than those of Experiment 4. In Phase One, the number of runs required for completion ranges from 250 to 750. In Phase Two, only 150 to 350 runs are needed. These results show that the controller requires more trials to complete Phase One than to complete Phase Two. Thus, for the task under study, the controller's required learning time depends on its prior task experience.



Initial Seed Value	Total trials necessary to reach 90% proficiency at random navigational task. Phase One	 Phase Two
0	250	150
1	400	150
2	650	200
3	750	200
4	400	150
5	250	350
6	350	300
7	350	250
8	300	300
9	250	350

Figure 12. Simulation Results for Experiment 5

Summary

To accommodate the aircraft's navigational mission, slight modifications were made in the controller's original design. The number of local demons was increased from 225 to 625. In addition, the demons' goal of maximizing force lifetimes was changed to minimizing force distances. Finally, a specialized reinforcement strategy was added to conclude each trial run. Despite these changes, the current controller still possesses the basic components that comprised the original design. Thus, the controller design offers flexibility in its application to simple aircraft tasks.

More important, the results of the two experiments demonstrate the controller's malfunction recovery capabilities. Although only one particular malfunction was studied, the controller's usefulness extends to others. Furthermore, the aircraft malfunction may occur at any instant instead of "waiting" for the controller to achieve task proficiency. This property stems from the fact that the malfunction conditions are transparent to the controller. It is important because real-life malfunctions occur unexpectedly.

The Controller in Perspective

The preceeding experiments describe the development of a controller to pilot a two-dimensional model aircraft. Now let us reflect on what the exercise has accomplished. Most significantly, it has provided a general framework for adaptive control that addresses the issue of malfunction recovery. Additionally, it demonstrates the controller's flexibility by applying it to two aircraft tasks. Finally, it provides an idea of the controller's tolerance to different initial conditions.

Though its effectiveness has only been studied with respect to a simplified aircraft model, this fact is of secondary importance (see footnote). Instead, the controller's primary importance derives from its capability to recover from malfunctions.

With these ideas in mind, let us examine the controller from a general perspective.

With the appropriate flight dynamics equations, and control actions corresponding to the actual deflections of an aircraft's control surfaces, the controller can be modified to pilot more advanced aircraft systems. The modification would entail changes in the problem space definition and local decision rules. However, the model's basic components and problem-solving strategy would remain unchanged.

Classical Control Systems

Although it features certain properties characteristic of a classical controller, the proposed controller is fundamentally different from a classical one. As with a classical controller, the proposed controller periodically outputs an actuating signal to the plant, or process, that it controls. However, in a classical controller, the actuating signals are pre-designed to correspond to different input states. In this sense, the classical controller "knows" *a priori*, the operating dynamics of the controlled process. In the proposed controller, the operating dynamics of the aircraft and its mission are not known beforehand. Instead, the controller must learn, by trial and error after the process begins, the correct actuating signals to issue for each aircraft state.

Another major difference involves the implementation of process feedback. In classical control systems, feedback takes on the form of an "error difference" between the plant's desired and actual performance. The controller uses this difference to adjust its output so that the error is reduced in subsequent plant execution. In the proposed control system, feedback has two forms, both of which differ from conventional methods. In the first form, feedback occurs only when the aircraft enters a success or failure state. This feedback signals the end of a trial run. Depending on the event (success or failure) that terminates the run, the controller adjusts its local decision rules. In the second form, feedback from the ACE "predicts" the aircraft's future performance based on a comparison between the current and previous aircraft states. The ASE uses this prediction to adjust the controller's logic for decisions leading to the current aircraft state. Consequently, this feedback influences the process only when a previous input pattern repeats itself. Thus, in one instance, feedback occurs infrequently and, in the other, its consequences do not occur immediately.

The primary reason for the controller's deviation from classical control theory arises from the objectives for its ultimate use. When initially configured, the controller can theoretically be provided with the exact operating dynamics of its plant. However, upon the occurrence of a plant malfunction, the plant's operating dynamics will change. Consequently, the controller's decision logic will no longer remain accurate. As such, the controlled process will fail unless the controller is designed to anticipate the particular malfunction conditions. Unfortunately, because of the unpredictable nature of most malfunctions, this capability is neither feasible nor practical. In this respect, a controller designed in the classical manner will not suffice. Instead, it is more desirable to design a controller capable of adapting to the conditions prevalent for its current plant configuration.

Adaptive Control

Depending on its context in this paper, the term "adaptive control" can take on two potentially confusing meanings. First, it can describe the process by which the controller learns to pilot the aircraft through its mission. Alternatively, it can describe the way the controller recovers the aircraft from a malfunction so that the aircraft can continue its mission. Both processes are related in the sense that the same control task must be accomplished though the plant configuration may vary. For this reason, subsequent references to adaptive control will convey its former, more common, meaning. As a final note, realize that the controller's malfunction recovery capabilites derive directly from the adaptive control method that it employs.

In general, as Truxal [11] explains,

> the primary interest in adaptive control lies in the possibilities of an automatic measurement of process dynamics and of an automatic and frequent redesign of controller characteristics.

These activities are present in the proposed controller. Until pre-established termination conditions are met, the controller continually measures the aircraft's position and velocity vectors. It uses these measurements to progressively modify its local decision rules with respect to an overall performance criterion. As a result, the controller is able to adapt to the aircraft's operating conditions in a manner that enables the aircraft's performance to improve.

Learning Systems

Because of its adaptive nature, the controller's task is not merely one of control itself; it is one of **learning to control.** Thus, to completely analyze the controller, one must consider its capacity for learning. Learning occurs by continually observing and tabulating the aircraft's performance. From these specific observations, the controller induces general conclusions as to the proper responses for different classes of input states. The learning process is then reflected in the manner in which the aircraft's measured performance improves with time.

As a machine learning paradigm, the controller exemplifies what Carbonell *et. al.* [3] call "learning from observation and discovery." However, a more precise classification comes from noting the functions of the ACE and ASE.

These adaptive logic elements in tandem provide what Widrow *et. al.* [12] call "learning with a critic." In this process, the controller learns its task via qualitative comparisons resulting from the application of an overall performance criterion to the outcome of its decisions.

Self-Organization

Implicitly related to the controller's adaptive control and learning capabilities is a desirable property known as "self-organization." Because the controller's design assumes no *a priori* knowledge of the aircraft's flight dynamics, the controller must learn its input-output decision logic from trial-and-error experience. As it accumulates flight dynamics information, the controller associates correct responses for each input state such that a map is created for the previously unknown problem space. Because the map is created *a posteriori*, the process of learning to pilot the aircraft is said to self-organize. For clarification purposes, Saridis [8] offers two definitions:

> *Self-Organizing Control Process:* A control process is called "self-organizing" if reduction of the *a priori* uncertainties pertaining to the effective control of the process is accomplished through information accrued from subsequent observations of the accessible inputs and outputs as the control process evolves.
>
> *Self-Organizing Controller:* A controller designed for a self-organizing control process will be called "self-organizing" if it accomplishes *on-line* reduction of the *a priori* uncertainties pertaining to the effective control of the process as it evolves.

A self-organizing controller is necessary as long as the actions governing the effective control of the given process are not provided from the outset. In the present context, this restriction arises because of the controller's intended use for aircraft malfunction recovery. Because of the unpredictable nature of malfunction situations, the particular conditions prevalent in a malfunction are difficult to anticipate. Therefore, it is desirable that the controller learn the particular conditions that apply to a given situation. As an added benefit, the controller can use experience gained in previous situations to accelerate its recovery time. In essence, self-organization renders the plant's operating conditions transparent to the controller.

Malfunction Recovery

The main result of this research has been the development of a controller that can recover in the event of a plant malfunction. This capability was demonstrated by the controller's performance in Experiments 4 and 5. As mentioned earlier, the controller's effectiveness generalizes to other malfunctions providing that the aircraft maintains enough directional control to fly its mission. For these reasons, the controller may be classified as a malfunction recovery system.

This classification does not give the controller any properties that have not already been discussed. Instead, it uniquely differentiates this controller from all others previously presented in the literature. Whereas other controllers have been designed with adaptive, learning, and self-organizing capabilities, their application has heretofore been limited to processes running under normal operating conditions. The present controller removes this restriction by operating effectively even after a plant malfunction. Because controlled processes are rarely immune to failure, controllers can only benefit from the incorporation of this capability.

Limitations of the Proposed Controller

A characteristic feature of self-organization involves the controller learning its task as the controlled process evolves. Because of this requirement, the controller's performance is highly dependent on the specificity of its feedback and the heuristics used to induce its control rules. Similarly, performance will vary depending on the selection of an appropriate state space. In light of these observations, the results reported here have not been optimal. Instead, they show that the controller can yield useful performance when applied to a non-trivial task.

As a malfunction recovery system, the controller requires that a solution exists for each malfunction situation. In this regard, its use is limited to controlled processes that exhibit "redundancy of control." When a unit fails, the controller withstands the failure by effecting compensating control actions from units still remaining operational. However, because of the redundancy of control requirement, more than one solution may exist for a given control task. Consequently, the controller may not always discover the "best" solution.

Conclusion

The research presented in this thesis shows how adaptive logic can be used to control a continuous process. In addition, it shows how a self-organizing controller can learn its task on-line. Self-organized learning is useful when only limited information is available *a priori,* as in the case of process malfunctions.

In conclusion, this thesis proposes a controller with two significant capabilities: (1) it can learn its task on-line; and (2) it can recover control even after a process malfunction. The first capability is not new; it can be found in controllers developed elsewhere in the literature. However, nowhere in the literature has a self-organizing controller been developed that addresses the issue of malfunction recovery. Herein lies the contribution of this work.

References

[1] Andrew, A.M. "Prerequisites of Self-Organization," in *Computer and Information Sciences,* J.T. Tou and R.H. Wilcox, Eds. (London: Cleaver-Hume Press, 1964) pp.381-91.

[2] Barto, A.G., R.S Sutton, and C.W. Anderson. "Neuron-Like Adaptive Elements That Can Solve Difficult Learning Control Problems," Coins Technical Report 82-20 (Amherst: University of Massachusetts Press) pp. 1-53.

[3] Carbonell, J.G., R.S. Michalski, and T. M. Mitchell. "An Overview of Machine Learning," in *Machine Learning: An Artificial Intelligence Approach,* Michalski, Carbonell, and Mitchell, Eds. (Palo Alto: Tioga Publishing Co., 1983) pp. 3-23.

[4] Donaldson, P.E.K. "Error Decorrelation: a Technique for Matching a Class of Functions," in *Proceedings of the Third International Conference on Medical Electronics,* (1960) pp. 173-78.

[5] Fu, K.S. "Learning Control Systems--Review and Outlook," *IEEE Transactions on Automatic Controls,* AC-15 (2), (1970) pp. 210-21.

[6] Michie, D., and R.A. Chambers. "BOXES: An Experiment in Adaptive Control," in *Machine Intelligence 2,* E. Dale and D. Michie, Eds. (Edinburgh: Oliver and Boyd, 1968) pp. 137-52.

[7] Michie, D., and R.A. Chambers. "'BOXES' as a Model of Pattern-Formation," in *Towards a Theoretical Biology;* 1, Prolegomena, C.H. Waddington, Ed. (Edinburgh: Edinburgh University Press, 1968) pp. 206-15.

[8] Saridis, G.N. *Self-Organizing Control of Stochastic Systems* (New York: Marcel Dekker, Inc., 1977).

[9] Savant, C.J., Jr. *Control System Design, 2nd Ed.* (McGraw-Hill, 1968).

[10] Schaefer, J.F., and Cannon, R.H., Jr. "On the Control of Unstable Mechanical Systems," in *Proceedings of the International Federation on Automatic Control 1966,* paper 6C.

[11] Truxal, J.G. The Concept of Adaptive Control," in *Adaptive Control Systems,* E. Mishkin and L. Braun, Jr., Eds. (McGraw-Hill, 1961) pp. 1-19.

[12] Widrow, B., N.K. Gupta, and S. Maitra "Punish/Reward: Learning with a Critic in Adaptive Threshold Systems," in *IEEE Transactions on Systems, Man, and Cybernetics,* Vol. SMC-3, No. 5, (Sept 1973) pp. 455-65.

[13] Widrow, B., and F.W. Smith "Pattern-Recognizing Control Systems," in *Computer and Information Sciences,* J.T. Tou and R.H. Wilcox, Eds. (London: Cleaver-Hume Press, 1964) pp. 288-317.

1. Report No. NASA CR-166620	2. Government Accession No.	3. Recipient's Catalog No.
4. Title and Subtitle An Application of Adaptive Learning To Malfunction Recovery		5. Report Date May 1986 6. Performing Organization Code
7. Author(s) *Robert Edward Cruz*		8. Performing Organization Report No. H-1325
9. Performing Organization Name and Address University of California, Los Angeles Los Angeles, California 90024		10. Work Unit No. RTOP 55-66-11 11. Contract or Grant No. NAG2-302
12. Sponsoring Agency Name and Address National Aeronautics and Space Administration Washington, D.C. 20546		13. Type of Report and Period Covered Contractor Report — Final 14. Sponsoring Agency Code

15. Supplementary Notes

NASA Technical Monitor: Eugene L. Duke, Ames Research Center, Dryden Flight Research Facility, Edwards, California 93523-5000

16. Abstract

A self-organizing controller is developed for a simplified two-dimensional aircraft model. The Controller learns how to pilot the aircraft through a navigational mission without exceeding pre-established position and velocity limits. The controller pilots the aircraft by activating one of eight directional actuators at all times. By continually monitoring the aircraft's position and velocity with respect to the mission, the controller progressively modifies its decision rules to improve the aircraft's performance. When the controller has learned how to pilot the aircraft, two actuators fail permanently. Despite this malfunction, the controller regains proficiency at its original task. The experimental results reported show the controller's capabilities for self-organizing control, learning, and malfunction recovery.

17. Key Words (Suggested by Author(s)) Expert systems Malfunction detection Real time		18. Distribution Statement Unclassified — Unlimited STAR category 63	
19. Security Classif. (of this report) Unclassified	20. Security Classif. (of this page) Unclassified	21. No. of Pages 43	22. Price* A03

**For sale by the National Technical Information Service, Springfield, Virginia 22161.*