

NPSCS-92-015

NAVAL POSTGRADUATE SCHOOL

Monterey, California

AD-A258 798



DTIC
ELECTE
JAN 6 1993
S B D

IMPROVING THE ASW SYSTEM EVALUATION TOOL

by

Yuh-jeng Lee

October 1992

Approved for public release; distribution is unlimited.

Prepared for:

Naval Postgraduate School
Monterey, California 93943

93-00322



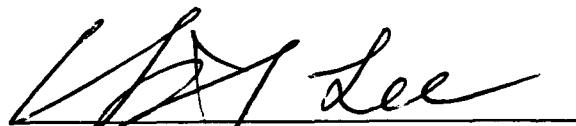
NAVAL POSTGRADUATE SCHOOL
Monterey, California

REAR ADMIRAL R. W. WEST, JR.
Superintendent

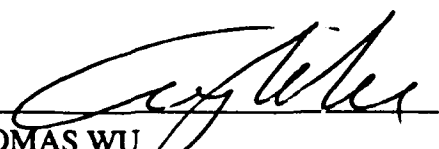
HARRISON SHULL
Provost

This report was prepared for and funded by the Antisubmarine Warfare Division (Op-71), Office of Chief of Naval Operations.

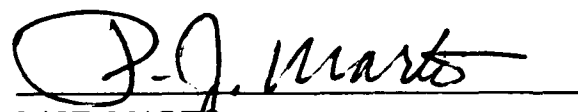
This report was prepared by:


YUH-JENG LEE
Assistant Professor of Computer Science

Reviewed by:


THOMAS WU
Associate Chairman for Research

Released by:


PAUL MARTO
Dean of Research

REPORT DOCUMENTATION PAGE

1a. REPORT SECURITY CLASSIFICATION UNCLASSIFIED			1b. RESTRICTIVE MARKINGS	
2a. SECURITY CLASSIFICATION AUTHORITY			3. DISTRIBUTION/AVAILABILITY OF REPORT Approved for public release; distribution is unlimited	
2b. DECLASSIFICATION/DOWNGRADING SCHEDULE				
4. PERFORMING ORGANIZATION REPORT NUMBER(S) NPSCS-92-015			5. MONITORING ORGANIZATION REPORT NUMBER(S)	
6a. NAME OF PERFORMING ORGANIZATION Computer Science Dept. Naval Postgraduate School		6b. OFFICE SYMBOL (if applicable) CS	7a. NAME OF MONITORING ORGANIZATION Naval Postgraduate School	
6c. ADDRESS (City, State, and ZIP Code) Monterey, CA 93943			7b. ADDRESS (City, State, and ZIP Code) Naval Postgraduate School Monterey, CA 93943-5100	
8a. NAME OF FUNDING/SPONSORING ORGANIZATION Antisubmarine Warfare Division		8b. OFFICE SYMBOL (if applicable) OP-71	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER	
8c. ADDRESS (City, State, and ZIP Code) Office of Chief of Naval Operations Washington, D.C. 20350-2000			10. SOURCE OF FUNDING NUMBERS	
			PROGRAM ELEMENT NO.	PROJECT NO.
			TASK NO.	WORK UNIT ACCESSION NO.
11. TITLE (Include Security Classification) Improving the ASW System Evaluation Tool				
12. PERSONAL AUTHOR(S) Yuh-jeng Lee				
13a. TYPE OF REPORT Final		13b. TIME COVERED FROM _____ TO _____		14. DATE OF REPORT (Year, Month, Day) October 1992
15. PAGE COUNT 18				
16. SUPPLEMENTARY NOTATION				
17. COSATI CODES			18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number)	
FIELD	GROUP	SUB-GROUP	ASW, System Evaluation Tool (ASSET), object-oriented design	
19. ABSTRACT (Continue on reverse if necessary and identify by block number)				
This report summarizes the work on the evaluation, design, and reimplementaion of part of the ASW System Evaluation Tool (ASSET), performed in the Computer Science Department, Naval Postgraduate School, under the sponsorship of the Antisubmarine Warfare Division (OP-71), Office of Chief of Naval Operations. We analyzed and implemented the improvements suggested in previous evaluations of various sub-areas of ASSET. In addition, we have designed and implemented a prototype user interface shell for ASSET on a Sun Sparcstation running X windows.				
20. DISTRIBUTION/AVAILABILITY OF ABSTRACT ☒ UNCLASSIFIED/UNLIMITED ☐ SAME AS RPT. ☐ DTIC USERS			21. ABSTRACT SECURITY CLASSIFICATION UNCLASSIFIED	
22a. NAME OF RESPONSIBLE INDIVIDUAL Yuh-jeng Lee			22b. TELEPHONE (Include Area Code) (408) 646-2361	22c. OFFICE SYMBOL CS/Lc

Improving the ASW System Evaluation Tool

Final Report

By

Yuh-jeng Lee

**Computer Science Department
U.S. Naval Postgraduate School
Code CS/LE
Monterey CA 93943**

ABSTRACT

This report summarizes the work on the evaluation, design, and re-implementation of part of the ASW System Evaluation Tool (ASSET), performed in the Computer Science Department, Naval Postgraduate School, under the sponsorship of the Antisubmarine Warfare Division (OP-71), Office of Chief of Naval Operations. We analyzed and implemented the improvements suggested in previous evaluations of various sub-areas of ASSET. In addition, we have designed and implemented a prototype user interface shell for ASSET on a Sun Sparcstation running X windows.

Implemented Improvements of the ASSET

1. Detection Model

A compound Lambda-Sigma jump detection model was implemented to simulate the detection of submarines. Also, the target's most detectable frequency and detection rate were allowed to vary with environmental region. Multiple engagements between platforms were allowed.

2. MPA Model

A glimpse rate model was used to determine detection opportunities of MPA and to approximate a continuous-looking sensor pattern. The glimpsing sensor field detects a target which is within the sensor region at the time of a glimpse with a probability of detection of 1.0.

ASSET allocates MPA to cues generated from the tracker-correlator by selecting SPA/MPA pairs using the ratio of MPA's time on-station to the SPA size as the selection criterion. If time permits, the MPA will stay on search region after prosecuting a submarine for another detection opportunity.

3. Converting ASSET code to CLOS

We investigated the feasibility and techniques of converting current ASSET code to CLOS. We used a conversion utility to help transform the ASSET code to CLOS. However, a significant portion of the modification has to be performed manually. Due to time constraints, only part of the conversion task was completed.

4. Reference

A detail report of the work mentioned above can be found in: P.-T. Chang, "Evaluation and Improvement of the ASW System Evaluation Tool", M.S. Thesis, Naval Postgraduate School, Monterey, California, March 1992.

Accession For	
NTIS GRA&I	☒
DTIC TAB	☐
Unannounced	☐
Justification	
By	
Distribution/	
Availability Codes	
Dist	Avail and/or Special
A-1	

DTIC QUALITY INSPECTED 8

Implementation of a User Interface Shell for ASSET on a SUN SPARCstation 2 running X Windows

A prototype User Interface was designed to allow similar interaction as given on the Macintosh computers. The goal was to allow for pull-down menus and pop-up dialog boxes in the X Window system. The only constraint was the system had to be written in Allegro Common Lisp.

To achieve the windowed nature of ASSET including the pull-down menus and pop-up dialog boxes, it was necessary to use a tool kit that would give that functionality. Based on the Common Lisp constraint, the Common Lisp Interface Manager (CLIM) was chosen as the tool kit.

What follows is a description of the code that is used in the user interface:

File: **asset-gui.cl**

Macro: **define-application-frame**

define-application-frame is a CLIM function that defines an application frame. This is used to specify the details of all panes that make up the user interface. Included in these details are the pane names and layout. For any particular pane, the pane type and various associated functions are specified. The following panes are used:

menu	pull down menu support
geoplot	all graphic drawing like maps will be drawn here the function draw-world is used to draw to the geoplot pane
alert	program generated comments are displayed here
avv	accepting values pane - this is the pop-up dialog manager
edit	the pane for normal user interacting with the system other than through dialog boxes - currently the introduction to the program is drawn here using the function draw-intro

The layout description specifies the proportions of pane sizes that will make up the user interface. See the accompany screen snapshot.

Method: **draw-intro**

This function draws the introductory screen to the **edit** pane of the application. The calls to **cursor-y-percent** merely position the following line of text vertically at that particular percentage of the total page size. **Center-text-in-window** centers each line of text in the **edit** pane. Both of these funtions together allow the text to be auto-centered and auto-placed independent of the window geometry.

Function: **cursor-y-percent**

Places the following text at the specified percentage down the edit pane. This function first queries the target stream for the overall height. It then sets the current cursor position to the proper value using the specified percentage.

Function: **center-text-in-window**

Centers the following text in the target window. This is accomplished using the available width from the target stream, along with the string width to determine the appropriate offset.

Macro: **define-asset-command (com-simulation-asset ...**

This macro describes the "Simulation" pull-down menu. In this macro, value is set based on the users selection. Once a selection is made, a case function calls the proper function for each selectable menu item. In this case only two menu items do anything, the "Edit Clock Parameters" selection and the "Quit" selection.

Macro: **define-asset-command (com-map-asset ...**

This macro describes the "Map" pull-down menu. The only active choice is "Edit Location/Size".

Function: **set-map-specs**

This function is responsible for displaying the pop-up dialog box to change the location or the size of the displayed map. Through the pop-up, the user can specify the current latitude, longitude and view radius.

Macro: **define-asset-command (com-blue-side-asset ...**

This macro describes the "Blue Side" pull-down menu. There are no active selections.

Macro: **define-asset-command (com-red-side-asset ...**

This macro describes the "Red Side" pull-down menu. There are no active selections.

Macro: **define-asset-command (com-utilities-asset ...**

This macro builds the "Utilities" menu and all associated sub-menus. There are no active selections.

Macro: **define-asset-command (com-workspace-asset ...**

This macro builds the "Workspace" pull-down menu. There are no active selections.

Macro: **define-asset-command (com-moe-asset ...**

This macro builds the "MOE" pull-down menu. There are no active selections.

Macro: **define-asset-command (com-choose-asset)**

This macro is currently called when the user selects "Edit Clock Parameters" from the "Simulation" pull-down menu. Upon selection a global variable is changed.

Macro: **define-asset-command (com-exit-asset)**

This macro is called when the user selects "Quit" from the "Simulation" pull-down menu.

Function: run-asset

After setting some pertinent variables, this function will create the top-level application frame for the interface and then run it.

File: drawing.cl

Function: set-default-drawing-globals

This function sets the latitude to 65 North and the Longitude to 10 West. It also sets the view radius of the map to 1500 km.

Method: draw-world

Calls the function draw map, that does the actual drawing.

Function: draw-map

Draw-map is responsible for scaling and drawing information to the geoplot pane. First, the window's inside height and width are determined. Next, the required spread in latitude and longitude for the specified view-radius is calculated. These values are view-lat and view-long respectively. Following this, the appropriate translations and scalings to make the map fit in the window are calculated. Finally, with the scaling and translations turned on inside of CLIM, the map is drawn.

In order to draw the map, the data was put into a list of polygons. The polygons are drawn one at a time. The polygons represent the coast-line of the continents and islands.

Following the map, the latitude and longitude lines are drawn and then the red dot that makes up the map center.

Function: draw-lat-long-lines

Currently the number of latitude and longitude lines to draw is determined by the magnitude of the view radius. If the view radius is greater than 8000 km, then latitude lines are drawn every 10 degrees and longitude lines are drawn every 20 degrees. If the view radius is between 3000 km and 8000 km, then the lines are drawn every 10 degrees for latitude and every 15 degrees for longitude. The final division is at a view radius of less than 3000 km. For this case, the latitude and longitude lines are drawn every 5 degrees and 10 degrees respectively.

Function: draw-map-center

A simple red circle marks the center of the map.

File: utils.cl

Function: create-asset-map

This function opens the raw map data file, and reads each polygon. Based on the size of the polygon, that is the number of points, the polygon is saved or discarded. This is a oversimplified filter that throws out the smaller polygons, so as to not clutter up the map.

File: `load-em.cl`

This file will load the three parts of the interface, namely `asset-gui.cl`, `new-map.cl`, and `drawing.cl`.

Running Asset:

To run asset, be on a machine that has Allegro Common Lisp and supports CLIM. Start Common Lisp, and execute `(load "load-em.cl")`. This will load all the pertinent files. After this, the root window must be created. To do this, execute `(setq *root* (clim:open-root-window :clx))`. Once the root window is created, ASSET can be run by: `(run-asset *root*)`.

Appendix: Source code for the User Interface Shell for ASSET on a Sun Sparcstation 2 running X windows

```

;;; File: asset-gui.cl

(setq *var* 1)

(clim:define-application-frame asset ()
  ()
  (:panes ((title :title
                  :display-string "New Asset")
            (menu :command-menu
                  :default-size :compute)
            (geoplot :application
                     :default-text-style '(:fix :bold :very-large)
                     :scroll-bars nil
                     :display-function 'draw-world)
            (alert :application
                   :default-text-style '(:fix :bold :very-large)
                   :scroll-bars nil)
            (avv :accept-values
                 :display-function '(clim:accept-values-pane-
displayer
                                     :displayer display-avv))
            (edit :application
                  :default-text-style '(:fix :bold :very-large)
                  :display-function 'draw-intro
                  :scroll-bars nil))))
  (:layout ((default
             (:column :rest
                      (menu :compute)
                      (:row :rest
                           (:column :rest
                                (geoplot :rest)
                                (alert 1/4))
                                (edit 1/3)))))))

(defmethod draw-intro ((frame asset) stream)
  (cursor-y-percent frame stream 10)
  (center-text-in-window stream "ASSET")
  (terpri stream)
  (cursor-y-percent frame stream 25)
  (center-text-in-window stream "ASW SYSTEMS")
  (terpri stream)
  (center-text-in-window stream "EVALUATION")

```

```

(terpri stream)
(center-text-in-window stream "TOOL")
(terpri stream)
(cursor-y-percent frame stream 60)
(center-text-in-window stream "Allegro CL")
(terpri stream)
(center-text-in-window stream "Build 0.1")
(terpri stream)
(cursor-y-percent frame stream 85)
(center-text-in-window stream "01 June 1992"))

(defun cursor-y-percent (frame stream percentage)
  (setq height (clim:window-inside-height (clim:get-frame-pane frame
'edit)))
  (multiple-value-bind (current-x-position current-y-position)
    (clim:stream-cursor-position* stream)
    (setq new-y-position (floor (* height (/ percentage 100))))
    (clim:stream-set-cursor-position* stream
      current-x-position
      new-y-position)))

(defun center-text-in-window (stream text)
  (setq avail-width (clim:stream-text-margin stream))
  (setq string-width (clim:text-size stream text))
  (clim:indenting-output (stream (floor (/ (- avail-width string-
width) 2))))
  (format stream text)))

(define-asset-command (com-simulation-asset :menu "Simulation")
  ()
  (setq value
    (clim:menu-choose
      '(("Edit Clock Parameters" :value 1)
        ("Edit Umpire Parameters" :value 2)
        ("Environmental Parameters ->"
          :item-list (("Enter Noise Freqs" :value 3)
                      ("Enter Proploss Curves" :value 4)
                      ("Define Environmental Regions" :value 5)))
        ("Start Simulation" :value 6)
        ("Abort Simulation" :value 7)
        ("Pause" :value 8)
        ("Continue" :value 9)
        ("Mode ->"
          :item-list (("Event Step" :value 10)
                      ("Time Step" :value 11)))
        ("Save to Disk" :value 12))

```

```

        ("Quit" :value 13))
      :label "Simulation"))
(case value
  (1 (com-choose-asset))
  (13 (com-exit-asset))))

(define-asset-command (com-map-asset :menu "Map")
  ()
  (setq value
    (clim:menu-choose
      '(("Edit Location/Size" :value 1)
        ("Refresh" :value 2)
        ("Plot Blue Tactical Picture" :value 3)
        ("Plot Red Tactical Picture" :value 4)
        ("Turn Display Off" :value 5))
      :label "Map"))
  (case value
    (1 (set-map-specs)))))

(defun set-map-specs ()
  (let ((stream (clim:frame-query-io clim:*application-frame*)))
    (clim:accepting-values (stream :own-window t :label "Map
Location / Size")
      (setq *lat* (clim:accept 'integer :default *lat*
                             :prompt "Latitude (Degrees)"
                             :stream stream))

      (terpri stream)
      (setq *long* (clim:accept 'integer :default *long*
                             :prompt "Longitude (Degrees)"
                             :stream stream))

      (terpri stream)
      (setq *view-radius* (clim:accept 'float :default *view-
radius*
                             :prompt "View Radius"
                             :stream stream))

      (terpri stream)
      (clim:accept-values-command-button (stream) "View Entire
World"

      (setq *full-screen-view* 1))))))

(define-asset-command (com-blue-side-asset :menu "Blue Side")
  ()
  (setq value
    (clim:menu-choose
      '(("Submarine" :value 1)
        ("Level 1 Command" :value 2))
    ))

```

```

("Level 2 Command" :value 3)
("ASWOC" :value 4)
("Fusion Center" :value 5)
("MPA Squadron" :value 6)
("Sub Op Auth" :value 7)
("Comm Sat" :value 8)
("Sensing Sat" :value 9)
("FAS Coverage Area" :value 10)
("SURTASS" :value 11)
("Surface Formation" :value 12)
("HFDF" :value 13)
("Mine Field" :value 14)
("Trip Wire" :value 15))
:label "Blue Side"))

(define-asset-command (com-red-side-asset :menu "Red Side")
  ()
  (setq value
    (clim:menu-choose
      '(("Submarine" :value 1)
        ("Level 1 Command" :value 2)
        ("Level 2 Command" :value 3)
        ("ASWOC" :value 4)
        ("Fusion Center" :value 5)
        ("MPA Squadron" :value 6)
        ("Sub Op Auth" :value 7)
        ("Comm Sat" :value 8)
        ("Sensing Sat" :value 9)
        ("FAS Coverage Area" :value 10)
        ("SURTASS" :value 11)
        ("Surface Formation" :value 12)
        ("HFDF" :value 13)
        ("Mine Field" :value 14)
        ("Trip Wire" :value 15))
      :label "Red Side")))

(define-asset-command (com-utilities-asset :menu "Utilities")
  ()
  (setq value
    (clim:menu-choose
      '(("Range Rings ->"
        :item-list (("Build" :value 1)
                     ("Plot" :value 2)
                     ("Erase" :value 3)
                     ("Delete" :value 4)
                     ("Plot Intersection" :value 5)

```

```

        ("Erase Intersection" :value 6)))
("Tracks      ->"
 :item-list (("Build" :value 7)
             ("Plot" :value 8)
             ("Erase" :value 9)
             ("Delete" :value 10)))
("Regions     ->"
 :item-list (("Build" :value 11)
             ("Plot" :value 12)
             ("Erase" :value 13)
             ("Delete" :value 14)))
("Plans       ->"
 :item-list (("Build" :value 15)
             ("Plot" :value 16)
             ("Erase" :value 17)
             ("Delete" :value 18))))
:label "Utilities"))))

(define-asset-command (com-workspace-asset :menu "Workspace")
  ()
  (setq value
    (clim:menu-choose
      '(("Blue ->"
        :item-list (("Recall" :value 1)
                    ("Save" :value 2)
                    ("Delete" :value 3)
                    ("Clear" :value 4)))
        ("Red  ->"
        :item-list (("Recall" :value 5)
                    ("Save" :value 6)
                    ("Delete" :value 7)
                    ("Clear" :value 8))))
      :label "Workspace"))))

(define-asset-command (com-moe-asset :menu "MOE")
  ()
  (setq value
    (clim:menu-choose
      '(("Display Blue Submarine Attrition" :value 1)
        ("Display Blue Surface Engagements" :value 2)
        ("Display Blue MPA Attrition" :value 3)
        ("Display Red Submarine Attrition" :value 4)
        ("Display Red Surface Engagements" :value 5)
        ("Display Red MPA Attrition" :value 6)
        ("Save MOEs" :value 7))
      :label "MOE"))))

```

```

(define-asset-command (com-choose-asset)
  ()
  (if (= *var* 1)
    (setq *var* 2)
    (setq *var* 1)))

(define-asset-command (com-exit-asset)
  ()
  (clim:frame-exit clim:*application-frame*))

(defun run-asset (root)
  (set-default-drawing-globals)
  (setq *earth-radius* 6378)
  (let ((asset (clim:make-application-frame 'asset
    :parent root :height 400 :width 700)))
    (clim:run-frame-top-level asset)))

;;; File:  drawing.cl

(setq *earth-radius* 6378)
(setq *full-screen-view* 0)

(defun set-default-drawing-globals ()
  (setq *lat* 65)
  (setq *long* -10)
  (setq *view-radius* 1500))

(defmethod draw-world ((frame asset) stream)
  (draw-map frame stream))

(defun draw-map (frame stream)
  ;;; determine the correct scaling to fit the world in the geoplot
  pane
  (let* ((height (clim:window-inside-height (clim:get-frame-pane
    frame 'geoplot)))
    (width (clim:window-inside-width (clim:get-frame-pane frame
    'geoplot)))
    (view-lat (* (* 2 (/ *view-radius* *earth-radius*)) (/ 180
    pi)))
    (view-long (* (* 2 (/ *view-radius*
    (* *earth-radius*
    (cos (* *lat* (/ pi 180)))))) (/
    180 pi)))
    ;;; calculate the transformations

```

```

        (scale-x (/ width view-long))
        (scale-y (* -1 (/ height view-lat)))
        (trans-x (- (/ view-long 2) *long*))
        (trans-y (* -1 (+ (/ view-lat 2) *lat*))))
    ;;; adjust the scales and translations for the window shape
    (cond ((>= width height)
        (setq scale-x (/ height view-long))
        (setq trans-x (- (/ (* view-long (/ width height)) 2)
            *long*)))
        ((> height width)
        (setq scale-y (* -1 (/ width view-lat)))
        (setq trans-y (* -1 (+ (/ (* view-lat (/ height width))
            2) *lat*))))))
    (clim:with-scaling (stream scale-x scale-y)
        (clim:with-translation (stream trans-x trans-y)
            (dolist (list *swdb*)
                (clim:draw-polygon* stream list :closed nil :ink
                    clim:+dark-sea-green+
                    :filled nil :line-thickness 2))
                (draw-lat-long-lines frame stream )
                (draw-map-center frame stream)))))

(defun draw-lat-long-lines (frame stream)
    ;;; the view-radius determines the number of lat-long lines to
    draw
    (cond ((>= *view-radius* 8000)
        (do* ((lat -80 (+ lat 10)))
            ((= lat 90))
            (clim:draw-line* stream -180 lat 160 lat :ink
                clim:+light-blue+))
        (do* ((long -180 (+ long 20)))
            ((= long 200))
            (clim:draw-line* stream long -90 long 90 :ink
                clim:+light-blue+)))
        ((> *view-radius* 3000)
        (do* ((lat -80 (+ lat 10)))
            ((= lat 90))
            (clim:draw-line* stream -180 lat 160 lat :ink
                clim:+light-blue+))
        (do* ((long -180 (+ long 15)))
            ((= long 195))
            (clim:draw-line* stream long -90 long 90 :ink
                clim:+light-blue+)))
        (t
        (do* ((lat -80 (+ lat 5)))
            ((= lat 85))

```



```

        (clim:draw-line* stream -180 lat 160 lat :ink
clim:+light-blue+))
        (do* ((long -180 (+ long 10)))
              ((= long 190))
              (clim:draw-line* stream long -90 long 90 :ink
clim:+light-blue+))))))

(defun draw-map-center (frame stream)
  (clim:draw-circle* stream *long* *lat* 1 :line-thickness 5 :ink
clim:+red+))

```

;;; File: utils.cl

```

(defun-map (filename)
  (with-open-file (output "asset-map2.cl" :direction :output
                        :if-exists :overwrite :if-does-not-exist :create)
    (format output "(setq *swdb*~%(list~%" )
    (with-open-file (input filename :direction :input)
      (let ((point-count 0)
            (polygon nil))
        (do ((item (read input nil input) (read input nil input)))
            ((eq item input))
          (cond ((= item 400)
                 (if (> point-count 80)
                     (output-polygon-to-file polygon output))
                 ;;; (format *standard-output* "~a~%" polygon)
                 (setq polygon nil)
                 (setq point-count 0))
                (t
                 (push item polygon)
                 ;;; (format *standard-output* "~a~%" polygon)
                 (setq point-count (+ point-count 1))))))))))

```

```

(defun output-polygon-to-file (poly file)
  (format file "'~a~%" poly))

```

;;; File: load-em.cl

```

(load "~/asset/asset-gui.cl")
(load "~/asset/new-map.cl")
(load "~/asset/drawing.cl")

```

Distribution List

Defense Technical Information Center Cameron Station Alexandria, VA 22304-6145	2
Dudley Knox Library Code 52 Naval Postgraduate School Monterey, CA 93943	2
Antisubmarine Warfare Division (OP-71) Office of Chief of Naval Operations Washington DC 20350-2000	1
Chairman, Computer Science Department Code CS Naval Postgraduate School Monterey, CA 93943	1
Prof. James N. Eagle Code OR/ER Department of Operations Research Naval Postgraduate School Monterey, CA 93943	1
Prof. Yuh-jeng Lee Code CS/Le Naval Postgraduate School Monterey, CA 93943	5