California AHMCT Research Center
University of California, Davis
California Department of Transportation

Automated Weed Detection and Spraying Phase II - (Implementation)

D.C. Slaughter
D.K. Giles
C.J. Tauzer
J. Speetjens

AHMCT Research Report

UCD-ARR-96-12-05-01

Final Report of Contract
IA65Q168 MOU 94-15

December 5, 1996

* This work was supported by the California Department of Transportation (Caltrans) Advanced Highway Maintenance and Construction Technology Center at UC Davis.

TABLE OF CONTENTS

LIST OF FIGURES

AHMCT GOAL STATEMENT

(i) "The research reported herein was performed as part of the Advanced Highway Maintenance and Construction Technology Program (AHMCT), within the Department of Mechanical Engineering, Aeronautical and Materials at the University of California, Davis and the Division of New Technology and Materials Research at the California Department of Transportation. Is evolutionary and voluntary. It is a cooperative venture of local, state, and federal governments and universities."

DISCLOSURE STATEMENT

(ii) Disclosure statement: The master agreement between the University and Caltrans requires the following disclaimer in or before the introduction.

"The contents of this report reflect the views of the author(s) who is (are) responsible for the facts and the accuracy of the data presented herein. The contents do not necessarily reflect the official views or policies of the STATE OF CALIFORNIA or the FEDERAL HIGHWAY ADMINISTRATION and the UNIVERSITY OF CALIFORNIA. This report does not constitute a standard, specification, or regulation."

(iii) Disclosure: A disclosure statement must be included in all published interim and final reports prepared for Caltrans by the contractor. The purpose of the statement is to identify the number and dollar amounts of all contracts or subcontracts in excess of $5,000 relating to the preparation of the report.

The contractor is free to copyright material, including interim reports and final reports, developed under the contract with the provision that Caltrans, and the FHWA reserve a royalty-free, non-exclusive and irrevocable license to reproduce, publish or otherwise use, and to authorize others to use, the work for government purposes.

INTRODUCTION

The California Department of Transportation expends a considerable amount of human and financial resources in its highway maintenance program for the control of vegetation along the shoulders of roadways. Effective weed control has multiple benefits including increased visibility and safety for drivers, reduced loss of natural resources (e.g. water) to unwanted vegetation and a reduction in alternative hosts for insect pests and diseases. Application of herbicides is one of the most effective methods for weed control.

A major issue in California is the current reliance on chemical methods of pest control. Helsel[1] estimated that in 1984 16 billion dollars were spent worldwide on pesticides. Further, Helsel reported the United States as the largest pesticide user in the world applying more than three times the quantity of pesticides as the number two country (Japan). Unfortunately, the continued reliance on chemically based pest control practices has potentially detrimental effects upon the environment and human health in the form of contamination of water supplies and soils. In addition, the effect of chemical residues is often cumulative and their continued use can be increasingly detrimental to the environment.

One approach for jointly maintaining effectiveness and predictability of herbicide weed control while reducing the amount of applied herbicide and potential environmental exposure is to increase the efficiency and selectivity of the mechanical process of spray application. Post-emergence herbicides are, by definition, only effective when deposited on living plant tissue. Deposition on soil, structural, paved or aggregate surfaces provides no weed control value and represents environmental contamination and a complete waste of resources. An ideal application system would deposit herbicide exclusively on living, target plants. Conceptually, such a system would detect the presence and location of living plant tissue and direct the appropriate amount of spray liquid to strike only the tissue.

The concept of intermittent spray control for plant sprayers has been previously investigated. Reichard and Ladd[2] discussed work in which plant conductivity and a charged probe were used to detect the presence of target plants. They also developed intermittent spray control systems which detected vegetable plants through spring steel wires and systems based on photodetectors. Season-long field tests of the control systems (Ladd et al.[3] and Ladd and Reichard[4]) found reduction of applied spray material to range from 24 to 51% with little or no reduction in pest control efficacy. The systems were physically limited to target plants which could fit between the sensor system since the basis of operation was the interruption of a light beam or the tripping of a wire switch.

Several researchers (e.g. Hollaender[5]) have documented the distinct absorption characteristics of chlorophyll which peak in the 675 nm region of the visible spectrum. Others have attempted to use this information to develop a non-contact sensor for detecting chlorophyll containing materials (e.g. plants) vs. non-chlorophyll containing materials (e.g. soils). For example, Hooper et al.[6] developed a photoelectric sensor that used a ratio of infrared to visible radiation under tungsten-halogen illumination to detect sugar beet, lettuce and cabbage seedlings in an automated thinning operation. This sensor, although fairly effective has the disadvantage of requiring tungsten-halogen illumination and is not well suited to the rugged environment of outdoor operations such as highway maintenance.

Haggar et al.[7] developed a hand-held weed sprayer that also used a ratio of infrared to visible radiation as a means of detecting vegetation. Even though the sensor developed by Haggar et al. was evaluated in a hand held operation where the operator carried the device while walking down the field, results indicated that the concept of using ratios of reflected light to detect vegetation was worth pursuing and could be developed into a practical implementation. Recently, two private companies have developed commercial prototypes of intelligent boom spraying systems that are based upon concept originally proposed by Hooper et al.

In an earlier phase of this project the concept of intelligent application of herbicides for roadside vegetation control was evaluated in a feasibility study. The system consisted of a real-time computer vision system which could detect live (green) plant material and was coupled to a set of rapid-response spray control valves and nozzles to permit selective application of herbicides to the detected plant material. The feasibility of this system was demonstrated and quantitative performance evaluations of spray deposition, system resolution and ground speed effects have been published in a previous report.

OBJECTIVE

The objective of this research is to build a second generation ruggedized spray control system extending the concept proven in an earlier phase of this research[8]. This second generation system will have increased system robustness in terms of physical durability and operational performance. It will be fabricated as a "field system", mounted and maintained on a Caltrans vehicle to remain in service and will be designed to spray weeds in a real-world setting.

METHODS AND MATERIALS

Similar to the earlier phase prototype the intelligent intermittent spray system (IISS) was developed from two fundamental elements: 1) a computer vision system, and 2) a rapid-response intermittent spray system. Shown in figure 1 is a photograph of the second generation system developed for this study.

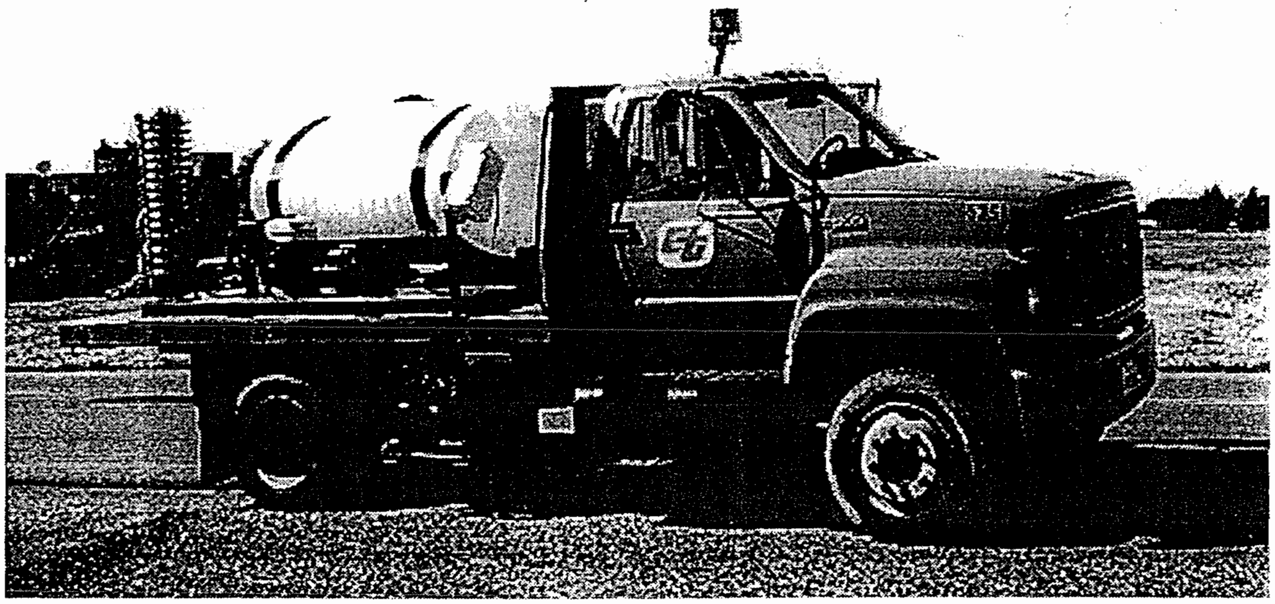

Figure 1. Photograph of the intelligent intermittent spray system.

This prototype was built on a platform which could be fitted onto a standard Caltrans 1.5 ton (GMC Topkick) flatbed truck. All the system components were chosen (whenever possible) to be suitable for actual highway shoulder weed control. The components and configuration of the intelligent intermittent spray system are described below.

Computer Vision System

A color computer vision system was developed to demonstrate the feasibility of detecting green plant material growing adjacent to the roadway. This system consisted of a solid-state color video camera (Sony, model SSC-C370), a computer video interface circuit board (RasterOps framegrabber, model 24MxTV), and a computer (Apple, PowerMacintosh 8100/80AV). The framegrabber was capable of digitizing true-color video images at a rate of fifteen frames per second. The resolution of the digitized color images was 640 columns by 480 rows. Once a video image was digitized it was stored in the memory of the framegrabber and could be accessed directly by the computer.

A color reference table was developed which could be used by the computer in real-time to determine if the color of a picture element (pixel) corresponded to one of several shades of green commonly observed in live plant material. The color reference table was stored in computer memory as a color look-up-table (CLUT) and was used by the computer to determine if any plant material was present in the field of view.

Each image was subdivided into sixteen regions of interest (ROI) where each ROI corresponded to a region of soil perpendicular to the right edge of the vehicle carrying the IISS, see figure 2. The IISS was equipped with sixteen spray nozzles and each of the sixteen image ROIs corresponded to one and only one of the sixteen spray nozzles. Each image ROI was defined as that portion of the computer image which contained the view of the soil which could be sprayed by its corresponding nozzle. For example, in figure 2 the ROI shown closest to the camera viewed the area of soil closest to the vehicle and this area of soil could be sprayed by the bottom spray nozzle. The size of the soil region represented by a single ROI in the direction perpendicular to the direction of vehicle travel was a function of the total width of right-of-way to be sprayed and the number of nozzles. The IISS was designed to spray a total right-of-way width of 8 feet and using 16 nozzles each ROI corresponded to a 6 inch (perpendicular to the direction of travel) region of right-of-way. The size of the soil region represented by each ROI in the direction of travel was controlled by the amount of time that each valve was held open at a given travel speed. The minimum size of a soil region corresponding to an individual ROI was an area approximately 6 inches in diameter. The resolution of the digitized image allowed the computer to identify plants as small as 0.124 square inches under ideal conditions. Under roadway maintanence conditions typically encountered by Caltrans the IISS can reliably detect plants 3.875 square inches and larger at the 8 mph maximum travel speed.

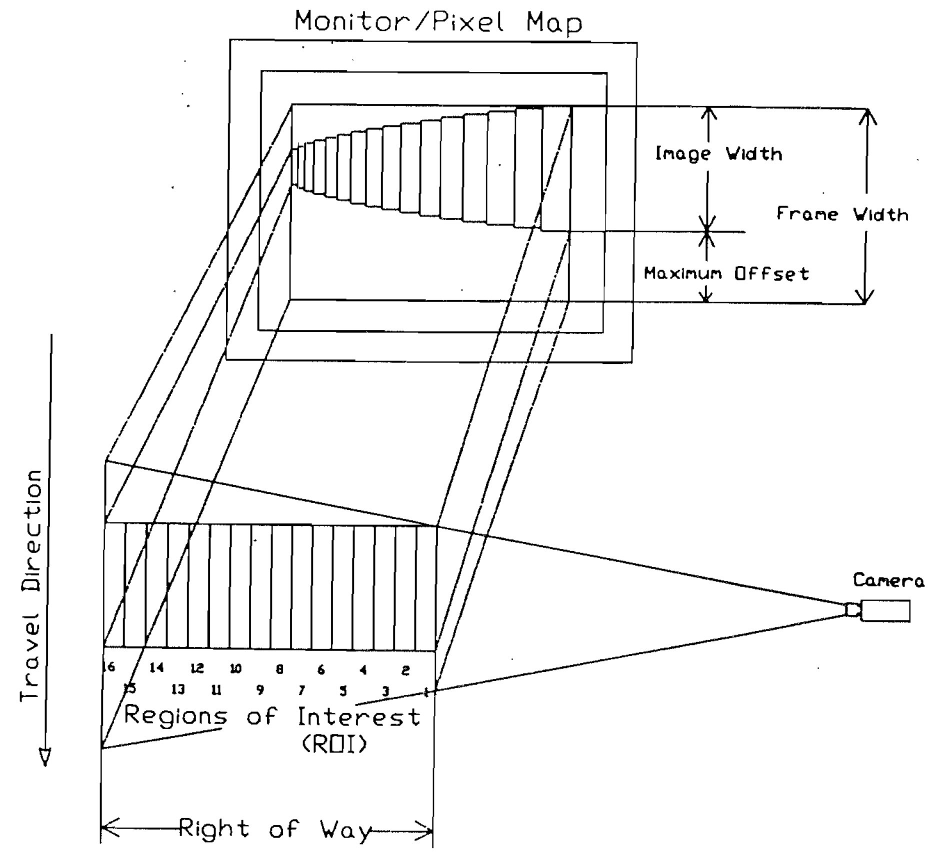

Figure 2. Schematic diagram showing the camera field of view and individual regions of interest.

The computer would examine each of the sixteen ROIs and determine which regions contained green plant material. If a region contained green plant material, a computer memory flag was turned on to indicate that the appropriate valve should be turned on in time to spray the plant detected in that region. The computer maintained a series of memory flags for each spray valve in a circular memory buffer and turned each spray valve on and off independently as needed to spray plant material in each region.

Rapid-Response Intermittent Spray System

A schematic diagram of the rapid-response intermittent spray system is shown in figure 3. The system consisted of a pump, filter, pressure regulator, manifold, sixteen solenoid valves, and sixteen spray nozzles. A manifold pressure of 40 PSI was required to provide a 0.42 gal/min flowrate through the spray nozzles. To maximize system response the nozzle was mounted very close to the exit port of the valve. To produce the desired spray patterns the lower 7 nozzles had a 15 degree spray dispersal pattern and the upper 9 nozzles had a 0 degree spray dispersal pattern. Each valve was controlled by the computer through a series of solid-state relays which allowed the computer to open or close each of the sixteen valves independently.

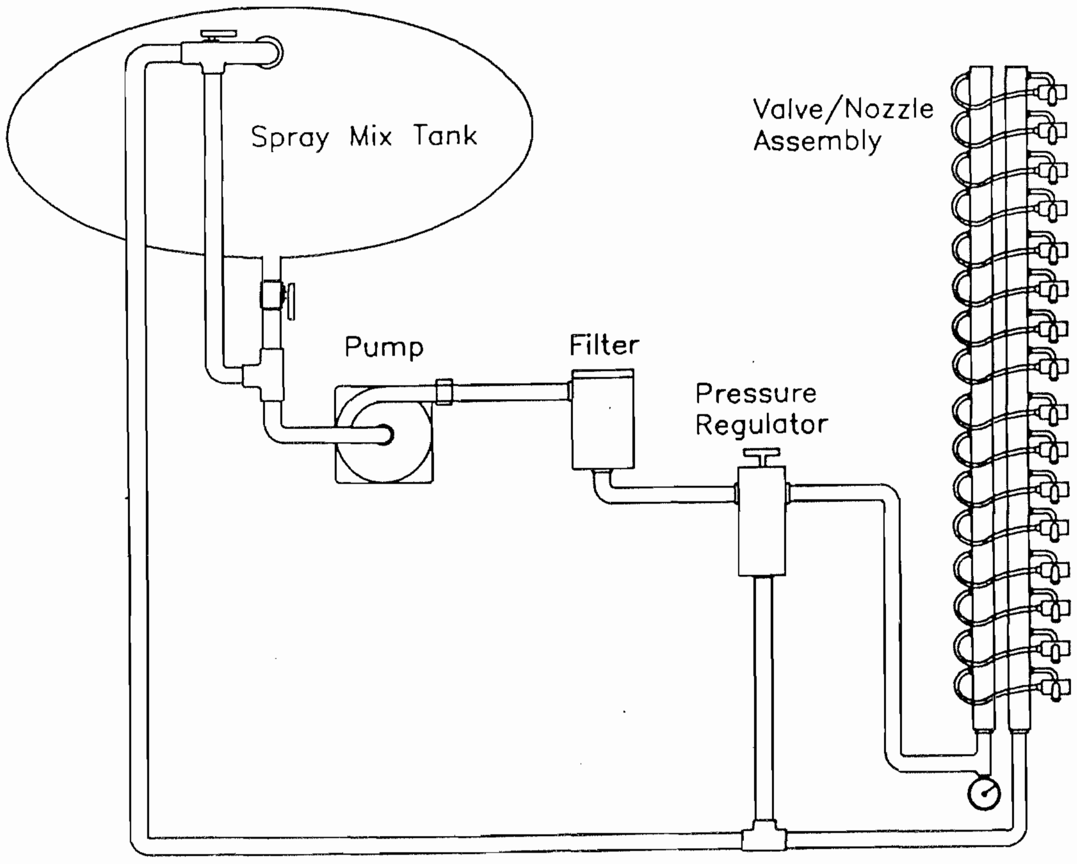

Figure 3. Schematic diagram of the rapid-response intermittent spray system.

Intelligent Intermittent Spray System Operation

A schematic diagram of the overall layout of the intelligent intermittent spray system (IISS) is shown in figure 4 and a flowchart showing the general sequence of events for operating the IISS is shown in figure 5. The sequence begins with a test to determine if the width of a spray region has been traveled by the system. The IISS used a radar displacement sensor (Raven Industries), mounted under the bed of the truck, to determine the distance traveled. This sensor emitted 130 electronic pulses per meter traveled and was capable of operating at travel speeds from 1.5 km/hr to 110 km/hr. Once the vehicle arrived at the new region to be analyzed the computer would begin the process of acquiring a new computer image of the new region. The acquisition of an image was conducted by the framegrabber as an independent task allowing the computer to monitor the travel of the vehicle and to control any of the sixteen valves as needed. Once the image was digitized and stored in the computer's memory analysis of the sixteen individual ROIs in the image was conducted. If any of the sixteen ROIs contained a weed, a flag was set in computer memory marking that ROI to be sprayed when the spray nozzle was positioned adjacent to the region of soil containing the weed. The computer used displacement information from the radar sensor to determine when the correct time to open each valve had occurred.

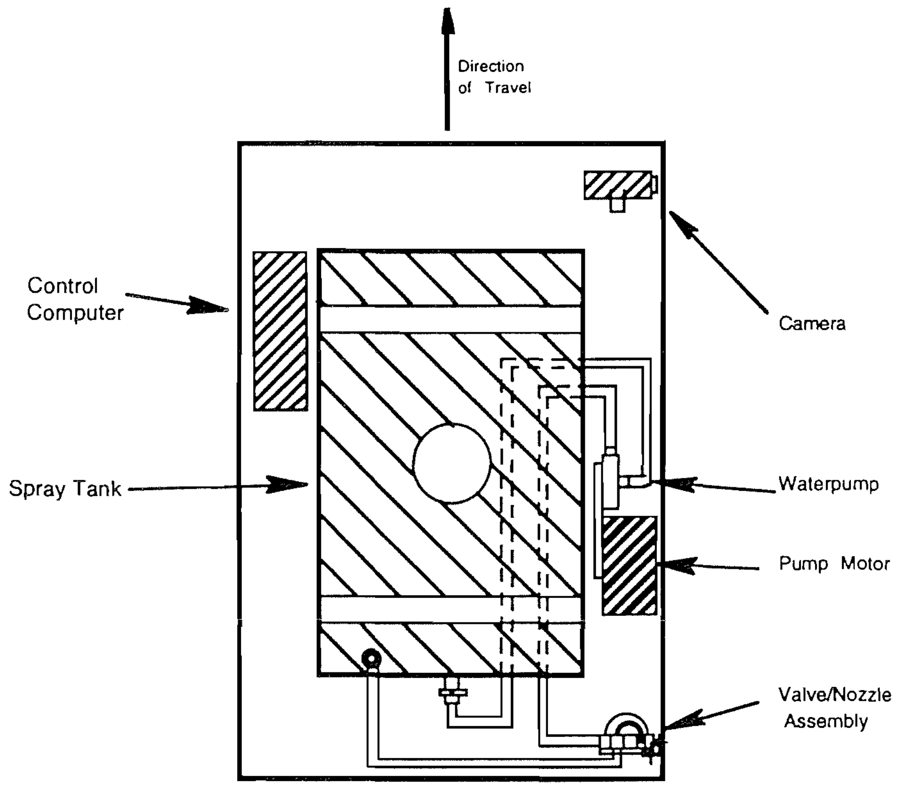

Figure 4. Schematic diagram (top view) of the prototype Intelligent Intermittent Spray System.

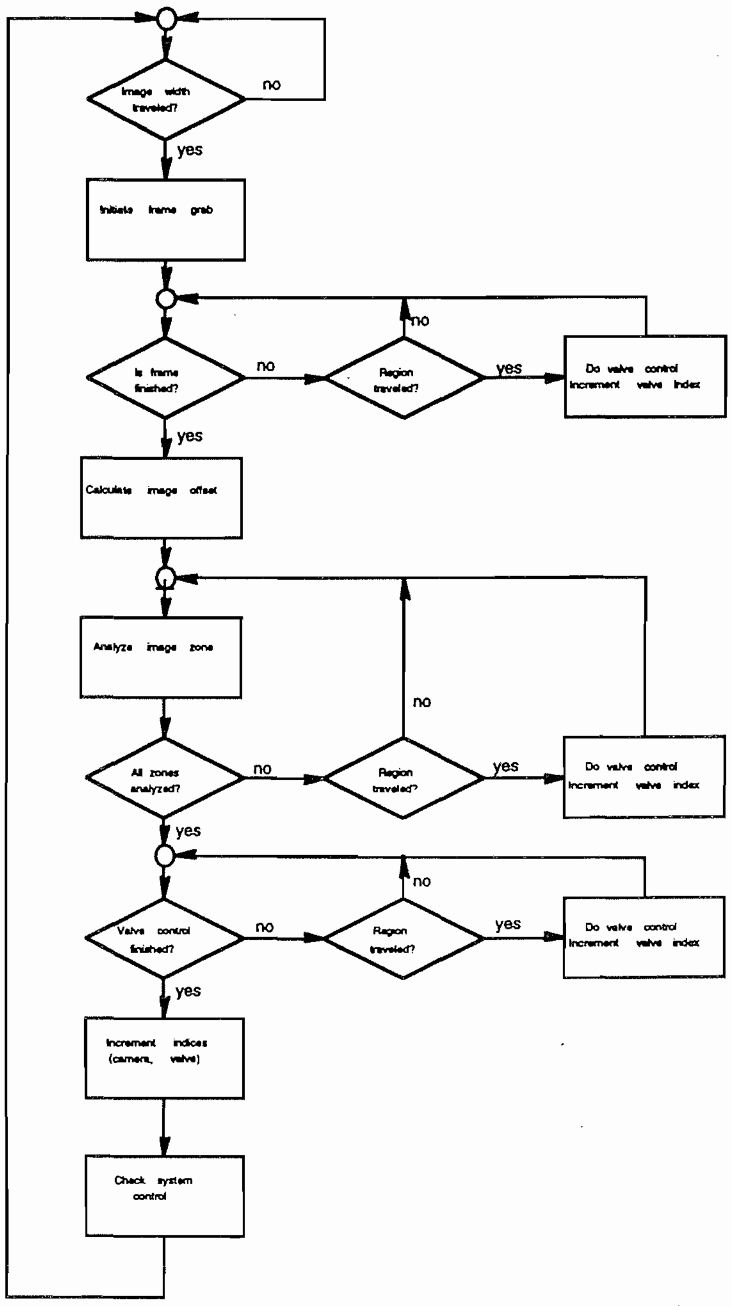

Figure 5. Flowchart showing the sequence of events for operating the IISS.

User Interface

The transition from a University development system to a field model to be used by Caltrans spraying crews in a real-world setting demanded a user interface which requires no previous knowledge of the system for operation. Therefore the user interface was limited to a simple control panel mounted inside the spray truck's cab. A drawing of this control panel and the system operating instructions that go with it can be seen in Figure 6.

To start the sprayer:

1 Make sure yellow plastic valves between tank and pump are open (both handles parallel with driving direction). Do not adjust brass regulator valve.

2 Start pump.

3 Make sure the master valve switch (A) is in off position (down). This inactivates all valves to prevent any unintentional spraying.

4 Turn main power switch (B) on. System is being started and is ready when green ready light comes on (takes approximately 3 minutes).

5 Select automatic or manual spraying mode (switch C).
- Automatic mode uses the automatic weed activated system.
- Manual mode allows user to manually control the valves.

6 The sixteen valve switches (D) control the individual valves. Up position is ready to fire in automatic mode and on in manual mode. The maximum spraying width is 8 feet starting 6 inches from the right side of the truck; valve number 1 sprays closest to the truck. For example the spraying width can be reduced to the first 4 feet next to the truck by switching off valves 9 thru 16.

Set 16 valve switches to desired spraying width.

7 The system is ready for operation but all valves are still inactivated by the master switch.

Turn on master valve switch (A). If in manual mode system will immediately start spraying! Maximum speed for use of the automatic weed activated system is 8 mph. At higher speeds the master switch must be turned off.

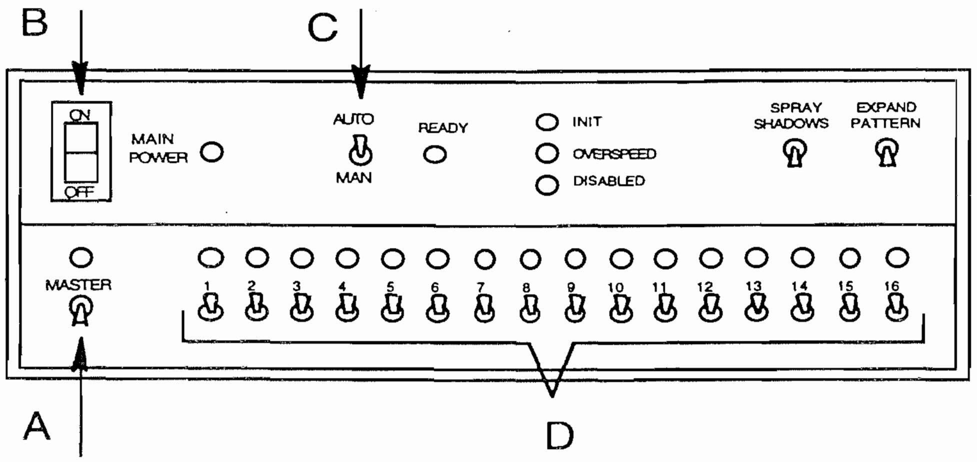

Figure 6. Layout of control panel and operating instructions.

List of IISS components

- Spraymix tank:	Raven 400 gallon tank
- Water pump:	Hypro model 9203C (max. 136 gpm, 180 psi, 6000 rpm)
- Pump drive:	Honda GX240 (242 cc, 8hp)
- Pressure regulator :	Spraying Systems Co. 110 P.R.
- Filter :	Spraying Systems Co. 1 1/4 NPT 150 psi max strainer
- Solenoids :	Kip Inc model 651064, 10 Watt, 12VDC
- Nozzles:	Spraying Systems Industrial VeeJet Series Nozzles, model 1505 for 7 lower positions, and model 0005 for 9 upper positions.
- Camera:	Sony SSC-C370, with autoiris lens.
- Inverter:	Tripp Lite PV550 Powerverter (550 W)
- Computer:	Apple Power Macintosh 8100/80AV
- Image processing board:	RasterOps 24MxTV, 24 Bit, 640x480 resolution
- Digital I/O board:	National Instruments CCA, LAB NB
- External disk drive:	Iomega Bernoulli Transportable 44 MB
- Disk for external drive :	Bernoulli Transportable 44 MB disk containing system software to run the computer system and the files required to run the intelligent spray system.
- Displacement Sensor:	Raven Industries, radar sensor.

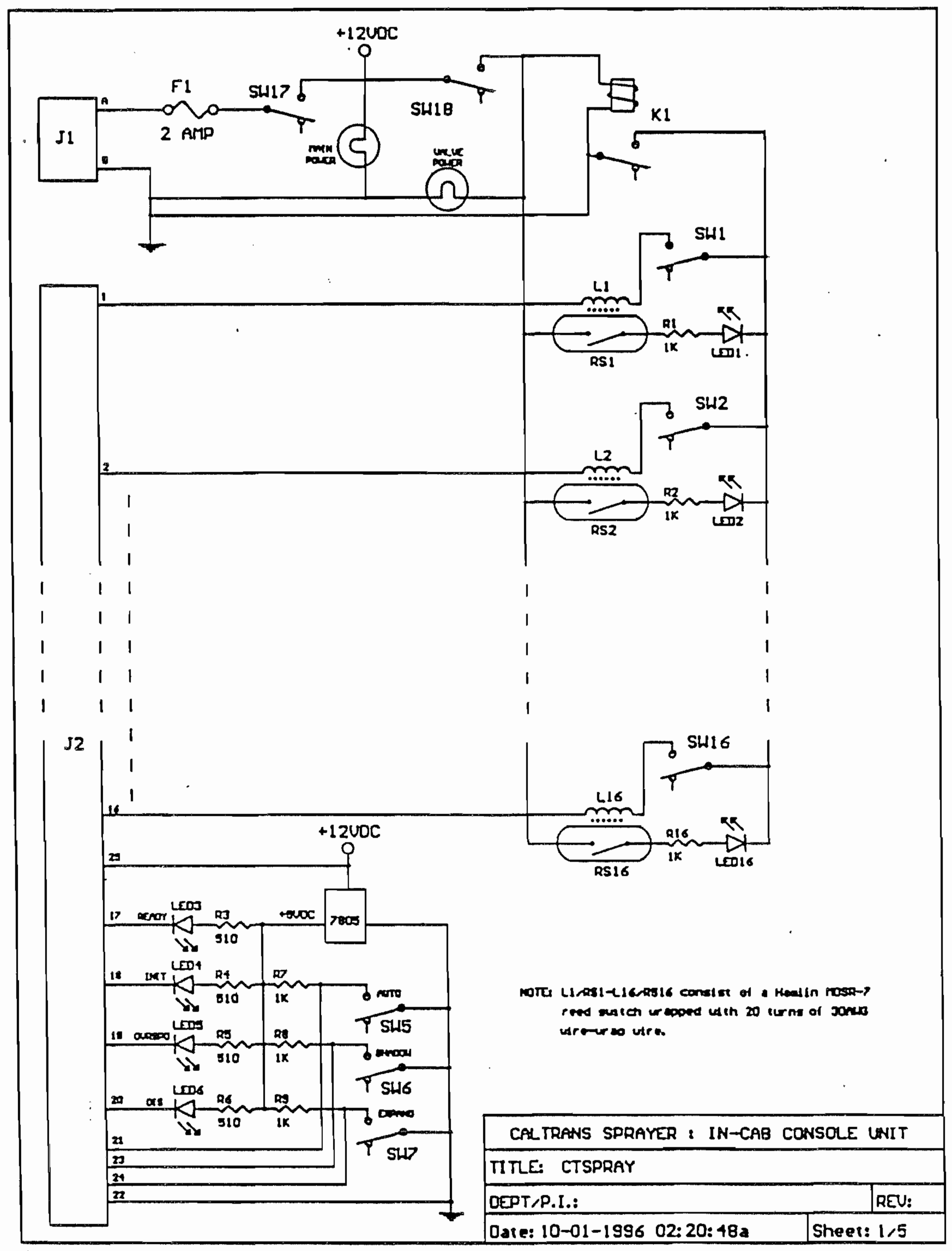

Figure 7. Wiring diagram for in-cab console unit.

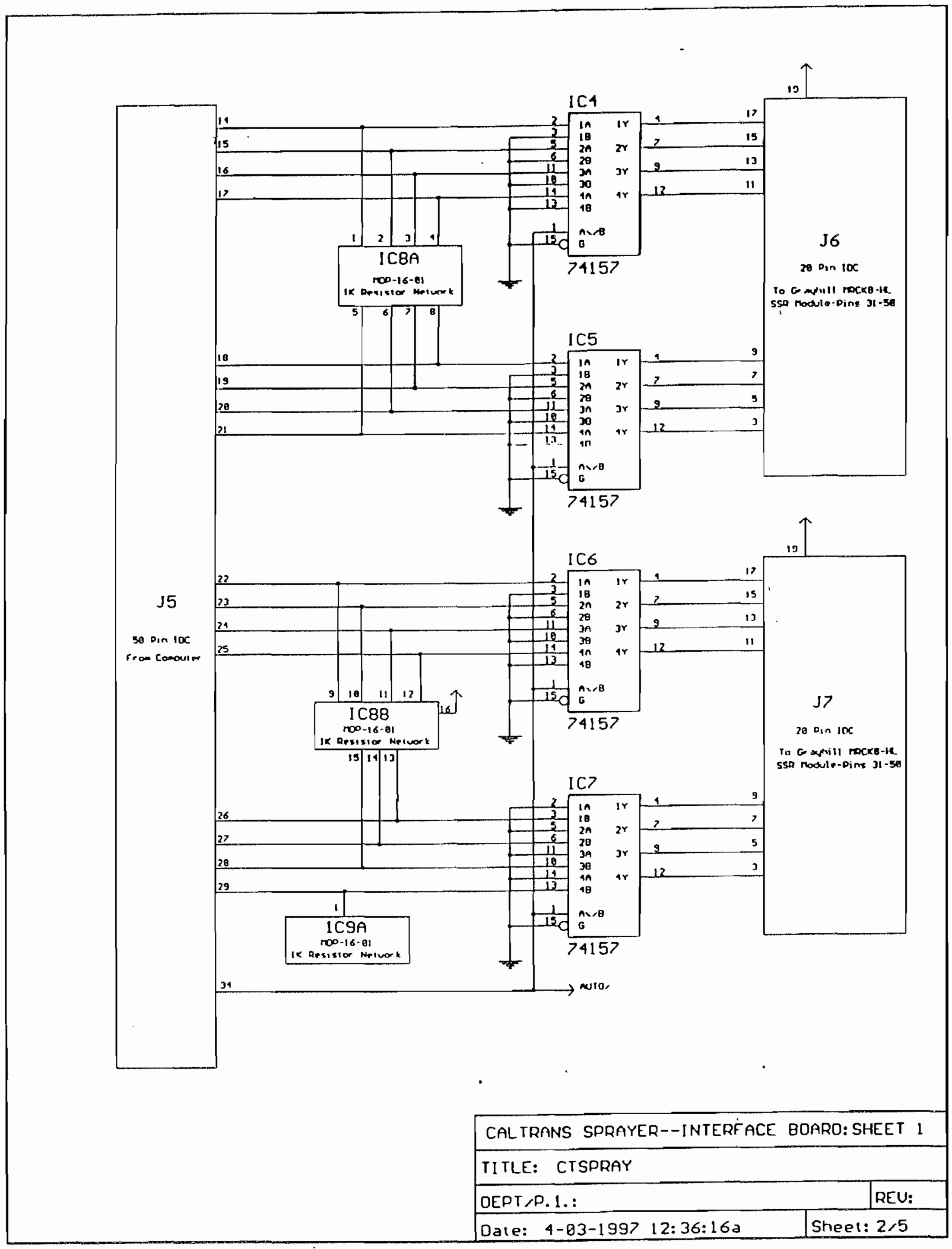

Figure 8. Wiring diagram for interface board (1).

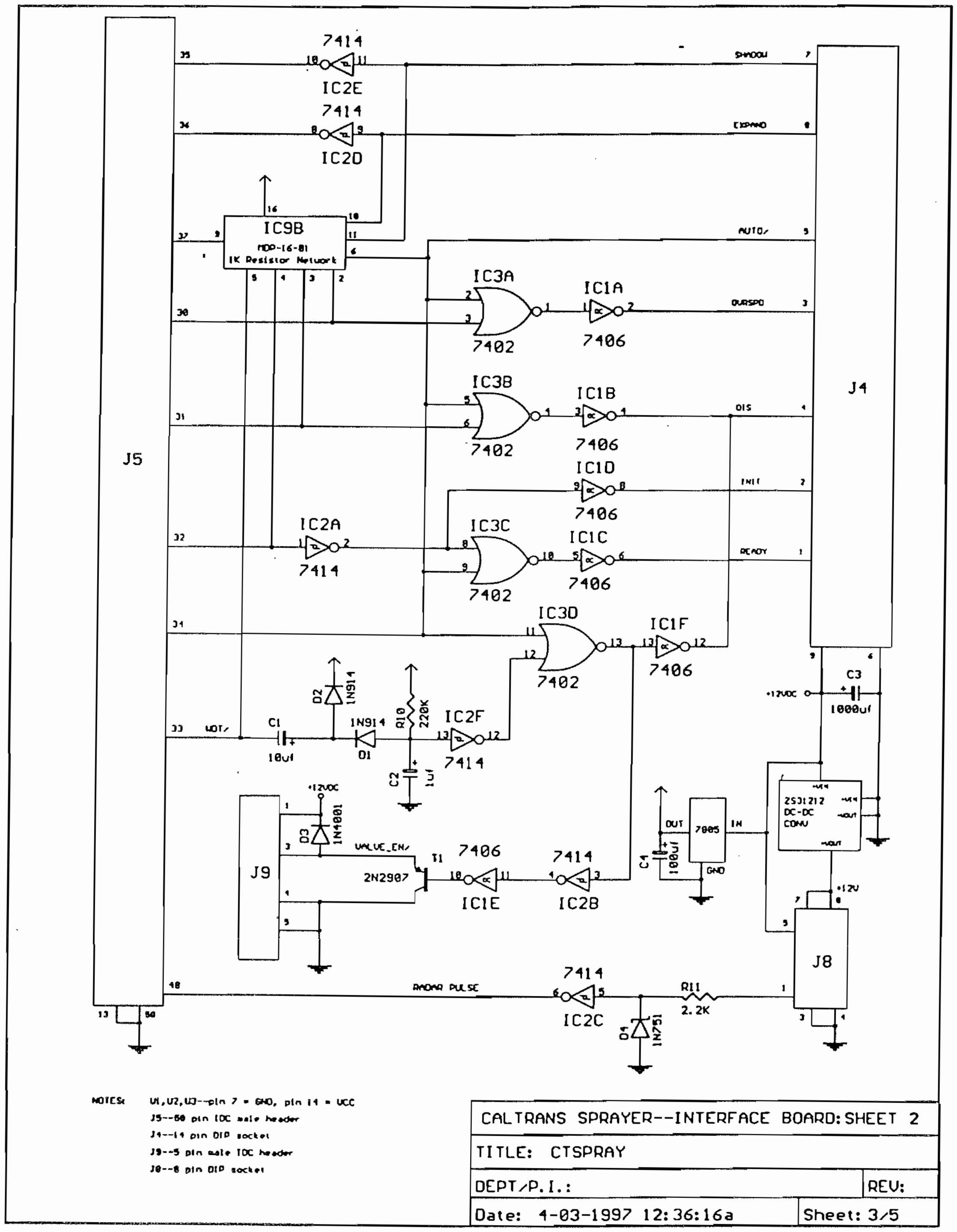

Figure 9. Wiring diagram for interface board (2).

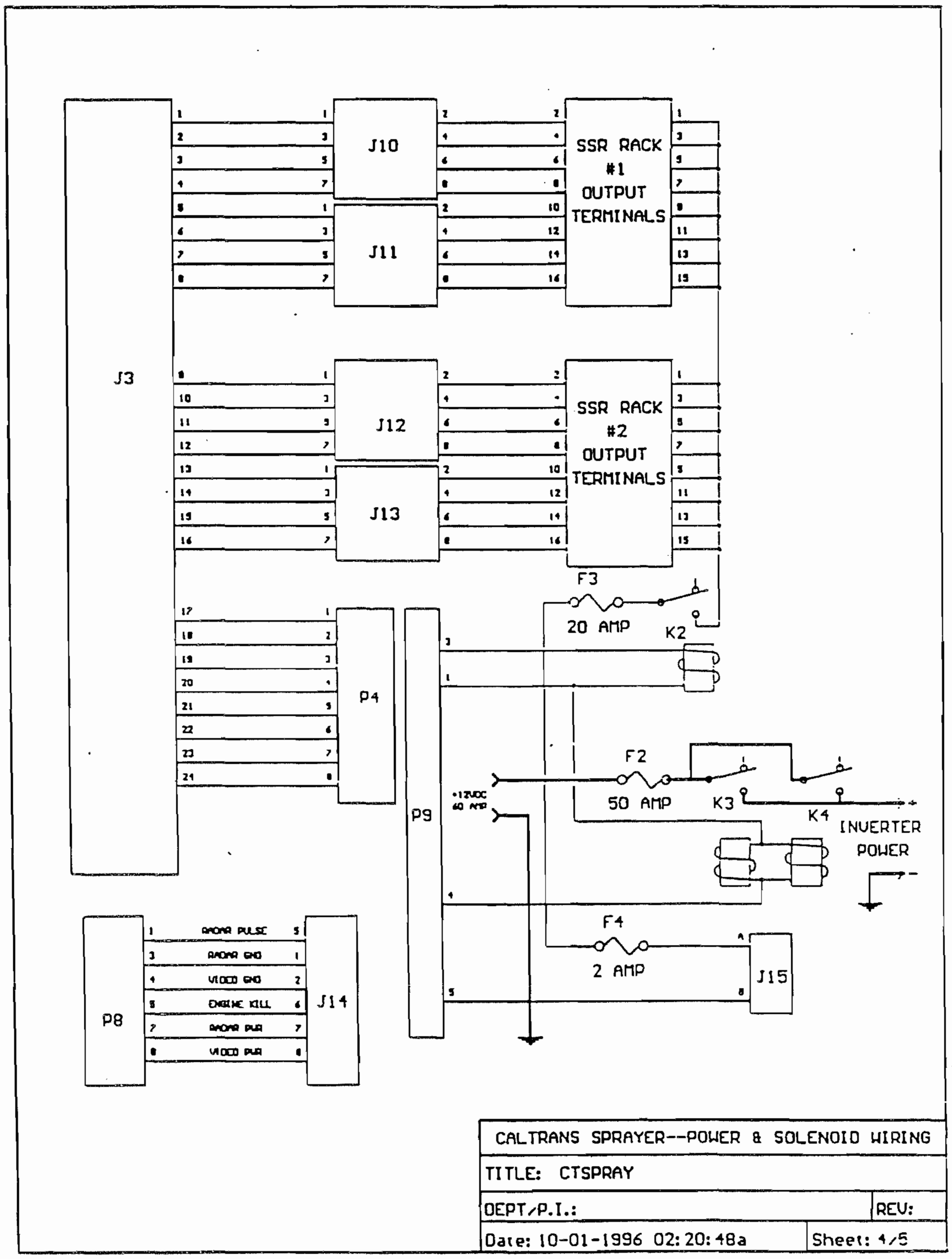

Figure 10. Schematic for power and solenoids.

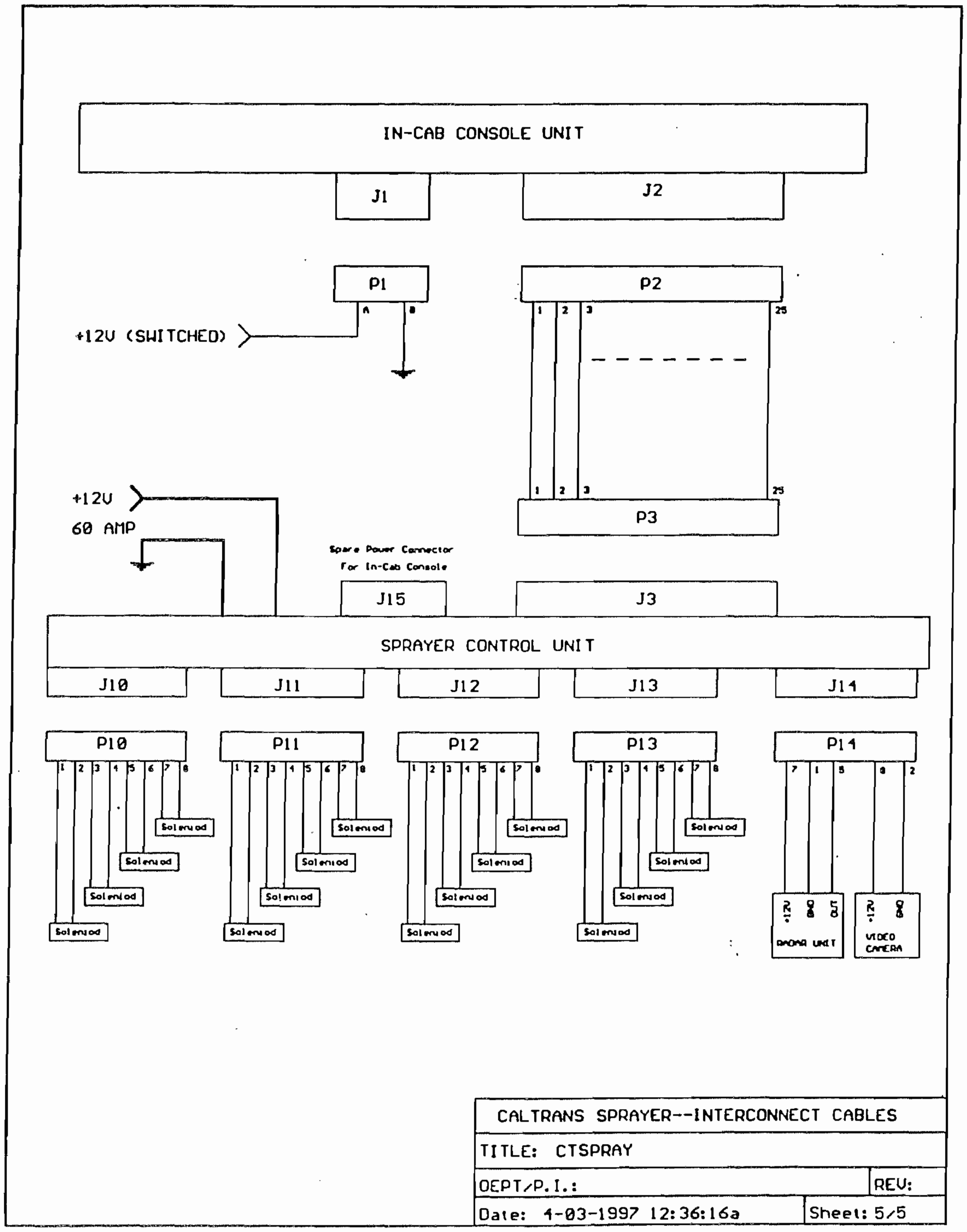

Figure 11. Schematic for interconnect cables.

TESTING AND PERFORMANCE

The fabrication of the field system was completed in the first quarter of 1996. Calibration of the system was conducted at that time, however due to the immediate need for system evaluation in actual Caltrans maintenance districts by Caltrans personnel no formal testing program was conducted at UC Davis. General observations of roadside operation on the UC Davis campus were conducted to verify that the performance was similar to the phase I prototype IISS. Performance of the phase I prototype system is documented in Slaughter et al.[8]

Feedback from Caltrans maintenance personnel indicated that the system generally performed well and that results were consistent with those of the phase I prototype. Some suggestions for improvement were: the need to reduce the erroneous spraying of roadway pavement containing black and white stones, possible reduction of maximum nozzle height to reduce spray drift, and a change in the tank/pump plumbing to allow tank to be completely emptied.

Based upon the observation of erroneous spraying of pavement containing black and white regions, a preliminary study was conducted at the UC Davis campus with a video camera similar to the one used in the phase II system and a second 3 CCD video camera. Computer images of black backgrounds with containing white objects were acquired with both of these two cameras. Analysis of these images showed that black and white boundary regions in images from the phase II camera contained highly saturated noise picture elements (pixels) some of which were bright green in color. Since these green noise pixels were of the correct color for weed material the system confused these boundary points for weeds and issued the command for spray activation. In the 3 CCD camera's images none of these colored noise pixels were found. It was hypothesized that in a single CCD camera, of the type used in the phase II system, black and white boundaries may occasionally partially illuminate a pixel causing a false reading of color. If a sufficient quantity of these black and white boundaries are present the system will incorrectly determine that there is a weed in the image. Because the green noise pixels that occur under these conditions are the same color as the color of actual weeds it is not possible to eliminate this problem by altering the system's color lookup table. It should however be possible to eliminate this problem by replacing the current phase II camera with one of higher quality which does not generate green noise pixels when viewing the boundary between black and white objects.

CONCLUSIONS AND RECOMMENDATIONS

This project successfully demonstrated the feasibility of automatic application of chemical herbicides to target weeds while minimizing the amount of chemical applied to non-target areas such as soil or roadway on a standard Caltrans 1.5 ton flatbed truck under real-world maintenance settings common to Caltrans weed control operations. These results indicate that the use of this technology could allow a significant reduction in the quantity of chemical herbicide applied by Caltrans with corresponding environmental and economic benefits.

Field tests of the technology under real-world maintenance settings common to Caltrans weed control operations illustrated the need for some additional developmental work to fully optimize system performance for use by Caltrans. First, the system currently uses a Macintosh computer to analyze images and control the spray activation. To increase the maximum operating travel speed it is recommended that the computer system be upgraded to include some hardware based color image analysis. A real-time color image processing subsystem such as the GPB color imaging hardware manufactured by Sharp Digital Information Systems and a high-speed MSDOS type computer would allow a significant increase in travel speed. The current IISS uses a removable Bernoulli disk drive for program and look-up-table storage. While this system is fairly rugged and has not failed during the 1996 or 1997 field tests, the use of a more stable solid-state memory source such as EPROM for program and look-up-table storage is recommended for a commercial prototype. The color video camera should be upgraded to reduce spray errors associated with noise in images of black and white roadway boundaries. Refinement of the color look tables should be conducted to improve the accuracy of the system under shadowy conditions or for weeds with different coloring than those common to the UC Davis campus.

REFERENCES

[1] Helsel, Z.R. 1987. Pesticide use in world agriculture. In: Z.R. Helsel (Ed.), Energy in plant nutrition and pest control. Elsevier, New York, p. 194.

[2] Reichard, D.L. and T.L. Ladd. 1981. An automatic intermittent sprayer. Trans. of the ASAE. Vol. 24(4):893-896.

[3] Ladd, T.L., D.L. Reichard, D.L. Collins and C.R. Buriff. 1978. An automatic intermittent sprayer: a new approach to the insecticidal control of horticultural pests. J. Econ. Entomology. Vol. 71:789-792.

[4] Ladd, T.L., and D.L. Reichard. 1980. Photoelectrically-operated intermittent sprayers for insecticidal control of horticultural pests. J. Econ. Entomology. Vol. 73:525-528.

[5] Hollaender, A. 1956. Radiation biology. Vol. III. McGraw-Hill. New York.

[6] Hooper, A.W., G.O. Harries, and B. Ambler. 1976. A photoelectric sensor for distinguishing between plant material and soil. J. Agric. Engng. Res. Vol. 21.

[7] Haggar, R.L. C.J. Stent, and S. Isaac. 1983. A prototype hand-held patch sprayer for killing weeds, activated by spectral differences in crop/weed canopies. J. Agric. Engng. Res. Vol. 28.

[8] Slaughter, D.C., D.K. Giles, and C.J. Tauzer. 1994. Intelligent Intermittent Spray System for Reduced Herbicide Control of Vegetation. AHMCT Research Report, UCD-ARR-94-08-01-01.

APPENDIX - IISS COMPUTER CODE

```
//-------------------------------------------------------------------------------------------------------------//
/                                                                                                               /
/    TAOS - v1.0                                                                                                /
/                                                                                                               /
/    Written By: Chris Tauzer                                                                                   /
/                November, 1995                                                                                 /
/                                                                                                               /
/    Hardware:     Power Macintosh w/NuBus                                                                      /
/                Raster-Ops video digitizer card (24MxTV compatible) - Slot B                                   /
/                National Instruments Lab-NB I/O card - Slot D                                                  /
/                                                                                                               /
/                                                                                                               /
/    Function: This program controls the offset sprayer developed by the Biological &                          /
/                Agricultural Engineering Department in conjunction with the                                    /
/                Caltrans AHMCT program.                                                                        /
/                                                                                                               /
/    Setup:    The video card needs to be set to 24 bit color (millions), 832 by 624                            /
/                                                                                                               /
//-------------------------------------------------------------------------------------------------------------//

// interface files
  #include <Windows.h>
  #include <Memory.h>
  #include <Events.h>
  #include <StdIo.h>
  #include <StdLib.h>
  #include <iostream.h>
  #include <math.h>
  #include <OSUtils.h>
  #include <ctype.h>
  #include <StandardFile.h>
  #include <Strings.h>
  #include <FCntl.h>
  #include <Files.h>
  #include <QuickDraw.h>
  #include <QDOffScreen.h>
  #include <Fonts.h>
  #include <time.h>

  #include "taos.h"              / general sprayer settings
  #include "taos_rsrc.h"         // menu etc. recource file
  #include "ROps_control.h"        // video digitizer functions
  #include "taos_utility.h"      // misc. useful functions
  #include "taos_im_analy.h"          // image analysis structures and functions
  #include "mac_ini.h"             // macintosh intialization
  #include "taos_valves.h"       // valve control structures and functions
  #include "taos_display.h"        // display functions (buttons, etc.)
  #include "taos_interface.h"    // keyboard, mouse, menu input functions

//----------------------------------------------------------------------------------------

// globals
  short          band_count;
  unsigned long  valve_action_increment;
  roi          zone[MAX_SET+1][MAX_BAND+1];
  float        band_offset_adj[MAX_BAND+1];
  short          valve_count;
  short        max_image_v = 403, max_offset;
  valves       valve[MAX_VALVE+1];
  WindowPtr        mainWindow;
  Rect         main_wrect;
  unsigned long plants[MAX_BAND+1] = {0};
  short          disp_status=0;

  short          set_count=0;
```

```
// system functions
  short vd_initialize(video_digitizer *vd);
  short get_roi_data(roi zone[MAX_SET+1][MAX_BAND+1], short *set_count, int control_increment,
        float band_adj[], short *band_count, unsigned long band);
  short make_rois(roi zone[MAX_SET+1][MAX_BAND+1], short *band_count, short *set_count,
        unsigned long base);
  void get_distance(unsigned long *distance);
  void do_image_adjust(short vd_number, short *brightness, short *contrast, short *hue,
        short *sat);
  void do_roi_adjust(void);
  void start_running(void);
  void stop_running(void);
  void do_show_lut(unsigned long base);
  void do_nice_exit(void);
  void show_plants(unsigned long plants[], int first_set, int last_set);
  void do_valves(valves valve[], unsigned long plants[], int speed, short call_index,
     int valve_action_increment);
  void send_valve_signal(unsigned long valve_signal);
  void show_valve_signal(unsigned long valve_signal);
  void show_speed(float speed);

  void do_image_mapping();

  void get_valve_data(valves valve[], short *valve_count);
  void update_roi_display(Rect *v_rect, Rect *h_rect, char *string);
  void save_settings(void);
  void do_valve_adjust();

  short set_video_display(video_digitizer *vd, short width, short height, Point start,
     short mode);

  void do_run_mode_control(unsigned char mode);
  void do_pattern_control(unsigned char mode);
  void do_shadow_control(unsigned char mode);

  extern void EventTask();

// ----- start of main function -----

int main(void)
{
  // variables
    unsigned char  din_c;
    char         string[100]={0};
    char         infile[20] = {0};

    int          set = -1, band = -1;
    int          err = 0, location=1;
    int          vnumber = 0;
    int          brightness, contrast, hue, saturation;

    short          call_index = 0;
    short        valve_allowance, fg_allowance;

    unsigned long fg_finish_time, fg_start_time, last_fg_start_time, fg_offset_ticks,
                 current_dist=0, fg_start_dist, fg_finish_dist, last_fg_start_dist,
                 next_fg_dist, next_valve_action_dist = 0, valve_signal=0;

    float        band_fg_offset[MAX_BAND+1], fg_offset;
    float        control_width;
    float        speed = 0.0, speed_sum=0.0, speed_now=0.0;

    FILE           *in_file;

    int loop = 0;
```

```
// set macintosh operating parameters
   initialize_mac();

// find vd slot / adddress
   if(err=openVideo(&vd[0].refnum, &vd[0].base))
      SysBeep(1);

// set up display window and misc.
   SetRect(&main_wrect, WINDOW_LEFT, WINDOW_TOP, 830, 620);
   mainWindow = NewCWindow(NULL, &main_wrect, "\pTarget Activated Offset Sprayer (TAOS)"
      , true, noGrowDocProc, (WindowPtr)(-1), false, NULL);
   initialize_display(1,1);
   draw_main_screen();

// get control width
   myprint(status_line, "Reading configuration file", 0);
   strcpy(infile, "taos.ini");
   if(in_file = fopen(infile, "r"))
   {
      while(strncmp(string, "CONTROL WIDTH", 13))
         fgets(string, 100, in_file);
      fscanf(in_file, "%f", &control_width);
      fclose(in_file);
   }
   else
   {
      sprintf(string, "File missing: &s", infile);
      myprint(status_line, string, 0);
      pause(5.0);
      exit(0);
   }

   valve_action_increment = control_width * RAVENS;  // convert meters to radar units
   valve_allowance = 0.2 * valve_action_increment;   // allow space before overspeed

// get image width
   if(in_file = fopen("taos.ini","r"))
   {
      while(strncmp(string, "IMAGE WIDTH", 11))
         fgets(string, 100, in_file);
      fscanf(in_file, "%hi", &max_image_v);
      fclose(in_file);
   }

// get image coefficients
   if(in_file = fopen("taos.ini","r"))
   {
      while(strncmp(string, "VD1 IMAGE COEFFICIENTS", 22))
         fgets(string, 100, in_file);
      fscanf(in_file, "%f", &vd[0].h_coef1);
      fscanf(in_file, "%f", &vd[0].h_coef2);
      fscanf(in_file, "%f", &vd[0].v_coef0);
      fscanf(in_file, "%f", &vd[0].v_coef1);
      fscanf(in_file, "%f", &vd[0].v_coef2);
      fclose(in_file);
   }

// get vd settings
   if(in_file = fopen("taos.ini","r"))
   {
      while(strncmp(string, "VD1 SETTINGS", 12))
         fgets(string, 100, in_file);
      fscanf(in_file, "%l", &brightness);
      fscanf(in_file, "%l", &contrast);
      fscanf(in_file, "%i", &hue);
      fscanf(in_file, "%i" , &saturation);
      fclose(in_file);
   }

   vd[0].brightness = (short)brightness;
   vd[0].contrast = (short)contrast;
   vd[0].hue = (short)hue;
   vd[0].saturation = (short)saturation;

   myprint(status_line, "Configuration finished", 0);

// initialize io board
   io_initialize();
   myprint(status_line, "I/O Initialized", 0);

// initialize video digitizer
   if(err = vd_initialize(&vd[0]))
   {
      myprint(status_line, "Video digitizer initialization error: ", err);
      exit(err);
   }
   myprint(status_line, "Video digitizer initialized", 0);

// read region information
   myprint(status_line, "loading roi data", 0);
   draw_video_area(ON);
   if(err = get_roi_data(zone, &set_count, valve_action_increment, band_offset_adj,
      &band_count, vd[0].base))
   {
      myprint(status_line, "roi file error: ", err);
      exit(err);
   }

// define maximum image offset allowed (ravens)
   max_offset =(vd[0].disp_rect.bottom - vd[0].disp_rect.top) - (set_count)
      * (zone[0][0].find_max_vpixel() - zone[0][0].find_min_vpixel());
   if(max_offset < 0)
      max_offset = 0;

// read valve information
   get_valve_data(valve, &valve_count);

// get color look-up table
   myprint(status_line, "Loading CLUT's", 0);
   if(err = get_lut())
   {
      SysBeep(60);
      myprint(status_line, "Color look up table file error: ", err);
   }

// adjust right side of video rectangle to minimum required
   draw_video_area(OFF);
   vd[0].image_rect.right = zone[0][band_count-1].find_max_hpixel();
   vd[0].width = vd[0].image_rect.right - vd[0].image_rect.left;// - 81;

   if(err=vd_set_rect(&vd[0].params, &vd[0].image_rect, vd[0].width, vd[0].height, 1))
   {
      myprint(status_line, "vd error: ", err);
      exit(err);
   }

// draw run screen
   initialize_display(band_count, valve_count);
   draw_video_area(ON);
   draw_data_area(ON);
   draw_over_speed_area(ON);
   draw_speed_area(ON);
```

```
    draw_controls(ON);

    fg_allowance = 0.1 * set_count * valve_action_increment;

//------------------------------------------------------------------------------------------

  // initial startup configuration
    status.show_pixels = NO;
    status.show_data = NO;
    status.run = NO;
    status.distance_counter = ON;

    myprint(status_line, "Initializing System", 0);

    get_distance(&current_dist);

    last_fg_start_dist = current_dist - 100;
    last_fg_start_time = TickCount()-60;
    next_fg_dist = current_dist;

    myprint(status_line, "System Ready", 0);
    write_dio(C, READY);

    pause(1.0);

// start loop  ---------------------------------------------------------------


  while(1)
  {
    // check user input, update system, wait here if not running

      EventTask();      // will loop here if not in automatic mode

      read_dio(C, &din_c);
      write_dio(C, din_c & ~OVER_SPEED);

    // initiate frame grab when we have traveled past the last image captured
      if(current_dist > next_fg_dist+fg_allowance)  // too far - gap in image info
        do_overspeed_warning(1);
      else
        while(current_dist < next_fg_dist)
          get_distance(&current_dist);

      grab_frame(1, vd[0].refnum, DONTWAIT);    // first slot with vd
      fg_start_time = TickCount();              // get frame grab start time
      get_distance(&fg_start_dist);             // get frame grab start location
      next_fg_dist = fg_start_dist + set_count * valve_action_increment;

    // re-initialize for new loop
      next_valve_action_dist = fg_start_dist + valve_action_increment;
      call_index = 0;

    // shift plant data for next image
      for(band = 0; band < band_count; band++)
        plants[band] = plants[band] << set_count;

    // calculate speed
      speed_now = ((float)(fg_start_dist - last_fg_start_dist)) / ((float)(fg_start_time -
        last_fg_start_time));     // (raven / ticks)

    // update frame grab start data
      last_fg_start_dist = fg_start_dist;
      last_fg_start_time = fg_start_time;

    // do a crude filtering
      if(speed_now < 20)
      {
        speed_sum -= speed;            // subract out current past average speed
        speed_sum += speed_now;        // add new instantaneous speed

        speed = speed_sum / 5.0;       // divide to get ravens per tick
      }

    // show speed, plant information available for the valve control
      if(status.show_data)
      {
        show_speed(speed);
        show_plants(plants, 0, 31);
      }

    // wait for frame to finish
      while(check_frame_grab(vd[0].refnum))     // 0=finished, 1=active one-shot
      {
        get_distance(&current_dist);
        if((current_dist >= next_valve_action_dist) && (call_index < set_count))
        {
          do_valves(valve, plants, speed, call_index, valve_action_increment);
          if(current_dist > (next_valve_action_dist+valve_allowance))
            do_overspeed_warning(0);       // too far - gap in spray coverage
          next_valve_action_dist += valve_action_increment;
          call_index++;
        }
      }

    // frame grab is finished - get time and location
      fg_finish_time = TickCount();
      get_distance(&fg_finish_dist);

    // analyze image

      // calculate point to start analysis (based on grab delay)
      fg_offset_ticks = fg_finish_time - fg_start_time - 2;    // ticks
      fg_offset = speed * fg_offset_ticks;                     // ravens

      // do each band
      for(band = 0; band < band_count; band++)
      {
        band_fg_offset[band] = band_offset_adj[band] * fg_offset; // pixels
        if(band_fg_offset[band] > max_offset)
        {
          band_fg_offset[band] = max_offset;       // not enough room to allow
          do_overspeed_warning(3);     // too much frame-grab delay - inaccurate
        }

        // do each set
        for(set = 0; set< set_count; set++)
        {
          // go do roi analysis
          if(zone[set][band].analyze_roi(band_fg_offset[band],
            status.show_pixels))
            plants[band] |= (1L << set);

          // check if valve control is needed before doing next roi
          get_distance(&current_dist);
          if((current_dist >= next_valve_action_dist)  && (call_index < set_count))
          {
            do_valves(valve, plants, speed, call_index, valve_action_increment);
            if(current_dist > (next_valve_action_dist+valve_allowance))
              do_overspeed_warning(0);    // too far - gap in spray coverage
            next_valve_action_dist += valve_action_increment;
            call_index++;  // number of valve controls so far this loop
```

```
                }
            }
        }

    // show results of image analysis
        if(status.show_data)
            show_plants(plants, 0, set_count-1);

    // finish up valve control before proceeding to next frame
        while(call_index < set_count)
        {
            get_distance(&current_dist);
            if(current_dist >= next_valve_action_dist)
            {
                do_valves(valve, plants, speed, call_index, valve_action_increment);
                if(current_dist > (next_valve_action_dist+valve_allowance))
                    do_overspeed_warning(0);    // too far - gap in spray coverage
                next_valve_action_dist += valve_action_increment;
                call_index++;
            }
        }
    }  // end of loop
}  // end of main()
//------------------------------------------------------------------------------------------------
// ----- start the control loop -----

void start_running()
{
    short band;

  // return if already in auto mode
    if(status.run)
        return;

  // update status line
    myprint(status_line, "system: running", 0);

  // clear plant information
    for(band = 0; band < band_count; band++)
        plants[band] = 0;

    if(status.show_data)
        show_plants(plants, 0, 31);

  // start running
    status.run = YES;
}
//------------------------------------------------------------------------------------------------
// ----- stop the control loop -----

void stop_running()
{
  short band;

  // return if already in manual mode
    if(!status.run)
        return;

  // turn valves off
    send_valve_signal(VALVES_OFF);

  // clear plant information
    for(band = 0; band < band_count; band++)
        plants[band] = 0;

  // update status line
    myprint(status_line, "system: stopped", 0);

  // stop running
    status.run = NO;
}
//------------------------------------------------------------------------------------------------
// ----- set up for mapping out the image -----

void do_image_mapping(void)
{
  // stop control loop
  status.run = OFF;
  send_valve_signal(VALVES_OFF);

  // do mapping
  draw_image_map_screen(ON);
  do_image_map(0);              // see taos_im_analy.h

  // redraw main display when finished
  draw_main_screen();
  draw_video_area(ON);
  draw_data_area(ON);
  draw_speed_area(ON);
  draw_over_speed_area(ON);
  draw_controls(ON);
}

//------------------------------------------------------------------------------------------------
// ----- display the current CLUT and allow it to be saved -----

void do_show_lut(unsigned long base)
{
  // stop control loop
  status.run = OFF;
  send_valve_signal(VALVES_OFF);

  // display the CLUT
  show_lut(base);               // see taos_im_analy.h

  // redraw main display when finished
  draw_main_screen();
  draw_video_area(ON);
  draw_data_area(ON);
  draw_over_speed_area(ON);
  draw_speed_area(ON);
  draw_controls(ON);
}
//------------------------------------------------------------------------------------------------
// ----- tidy up before quitting (not used from operator console) -----

void do_nice_exit(void)
{
  short vd_number = 0;
  WindowPtr current_window;

  // tidy up before quitting
    send_valve_signal(VALVES_OFF);
    myprint(status_line, "Resetting video digitizer", 0);
    vd_reset(&vd[vd_number].params);
    while(current_window = FrontWindow())
        DisposeWindow(current_window);

    exit(0);
}
//------------------------------------------------------------------------------------------------
// ----- read the counter to get the total number of radar pulses since startup -----
```

```
void get_distance(unsigned long *count)
{
static unsigned long last_count = 0, overflow_record = 0;

   unsigned char din_c;
   //static unsigned long lastTime = TickCount();

   char string[20] = {0};

   // first let's use this opportunity get watchdog ready for a pulse
      read_dio(C, &din_c);
      write_dio(C, din_c & ~WATCHDOG);

   // get the new distance count

      // read the radar pulses
      read_counter(B2, count);      // see lab_nb_control.h

      // increment overflow record if counter has started over again
      if(*count < last_count)
         overflow_record++;

      // remember where we are to check progress next time
      last_count = *count;

      // this is the total distance traveled in radar pulses (130/meter)
      *count += overflow_record * 65536;

   // check for control input (not moving?)
      if(*count == last_count)
         EventTask();

   // set watchdog up for next kick
      write_dio(C, din_c | WATCHDOG);

}
//---------------------------------------------------------------------------------
// ----- valve control function -----

void do_valves(valves valve[], unsigned long plants[], int speed, short call_index,
   int valve_action_increment)
{
   long valve_signal = 0;
   short vnumber;

   for(vnumber=0; vnumber < MAX_VALVE; vnumber++)
   {
      if(valve[vnumber].calculate_valve_signal(plants, speed, call_index,
         valve_action_increment))
         valve_signal |= (1 << vnumber);    // see taos_valves.h
   }
   send_valve_signal(valve_signal);

   if(status.show_data)
      show_valve_signal(valve_signal);
}
//---------------------------------------------------------------------------------
// ----- send valve signal to relays -----

void send_valve_signal(unsigned long valve_signal)
{
   // send low 8 bits to port a, next 8 to port b (16 more for expansion)
      write_dio(A, valve_signal);
      write_dio(B, valve_signal>>8);
}
```

```
//---------------------------------------------------------------------------------
// ----- set valve control mode (off, manual on, auto) -----

void do_valve_control(Point mouseloc)
{
   short v;
   unsigned char din_a, din_b;
   unsigned short current_valve_control= 0x0000;

   read_dio(A, &din_a);
   read_dio(B, &din_b);
   current_valve_control = ((din_b) << 8) | (din_a);

   for(v=0; v<MAX_VALVE; v++)
   {
      // set specific valve to auto mode
      if(mouse_in(valve_auto_rect[v], mouseloc))
      {
         valve[v].set_status(AUTO);
         current_valve_control &= ~(1 << v);
         send_valve_signal(current_valve_control);
         display_valve_control(v, AUTO);
         show_valve_signal(current_valve_control);
         break;
      }

      // turn specific valve on
      if(mouse_in(valve_on_rect[v], mouseloc))
      {
         valve[v].set_status(MANUAL_ON);
         current_valve_control |= (1 << v);
         send_valve_signal(current_valve_control);
         display_valve_control(v, MANUAL_ON);
         show_valve_signal(current_valve_control);
         break;
      }

      // turn specific valve off
      if(mouse_in(valve_off_rect[v], mouseloc))
      {
         valve[v].set_status(MANUAL_OFF);
         current_valve_control &= ~(1 << v);
         send_valve_signal(current_valve_control);
         display_valve_control(v, MANUAL_OFF);
         show_valve_signal(current_valve_control);
         break;
      }

      // set all of the valves to auto mode
      if(mouse_in(all_auto_rect, mouseloc))
      {
         for(v=0; v < MAX_VALVE; v++)
         {
            valve[v].set_status(AUTO);
            send_valve_signal(0x0000);
            display_valve_control(v, AUTO);
            show_valve_signal(0x0000);
         }
         break;
      }

      // turn all of the valves on
      if(mouse_in(all_on_rect, mouseloc))
      {
         for(v=0; v < MAX_VALVE; v++)
         {
                valve[v].set_status(MANUAL_ON);
                send_valve_signal(0xffff);
                display_valve_control(v, MANUAL_ON);
                show_valve_signal(0xffff);
            }
            break;
        }

        // turn all of the valves off
        if(mouse_in(all_off_rect, mouseloc))
        {
            for(v=0; v < MAX_VALVE; v++)
            {
                valve[v].set_status(MANUAL_OFF);
                send_valve_signal(0x0000);
                display_valve_control(v, MANUAL_OFF);
                show_valve_signal(0x0000);
            }
            break;
        }
    }
}

//----------------------------------------------------------------------------------
// ----- display results of plant detection -----

void show_plants(unsigned long plants[], int first_set, int last_set)
{
    Rect plant_rect;
    int left, top;
    int band, set;
    int width = 8, height = 8, skip = 1;

    left = data_rect.left - (width+1) + 1;

    for(band = 0; band < band_count; band++)
    {
        left += width + 1;
        top = data_rect.top + (first_set-1) * (height + 1);

        for(set = first_set; set <= last_set; set++)
        {
            top += height + 1;
            SetRect(&plant_rect, left, top, left+width, top+height);
            if(plants[band] & (1L << set))
                my_fill_rect(&plant_rect, 0x0000ff00, skip);
            else
                my_fill_rect(&plant_rect, 0x00555566, skip);
        }
    }
}

//----------------------------------------------------------------------------------
// ----- show valve control signal -----

void show_signal(char band, char front_plant, char back_plant, unsigned long color)
{
        Rect signal_rect;
        int left, top;
        char set;

        int width = 8, height = 8, skip = 2;

        left = data_rect.left + (width + 1) * band+ 1;

        for(set = front_plant+1; set <= back_plant+1; set++)
        {
            top = data_rect.top + set * (height + 1);
            if(top < data_rect.top || top > data_rect.bottom)
                return;
            SetRect(&signal_rect, left, top, left+width, top+height);
            my_fill_rect(&signal_rect, color, skip);
        }
}

//----------------------------------------------------------------------------------
// ----- show valve signal -----

void show_valve_signal(unsigned long valve_signal)
{
        Rect signal_rect;
        int left, top, bottom;
        char v;

        int width = 11, height = 8, skip = 1;

        top = valve_rect.top;
        left = valve_rect.left - (width-1);
        bottom = valve_rect.bottom;

        for(v = 0; v < MAX_VALVE; v++)
        {
            left += width+1;
            SetRect(&signal_rect, left, top, left+width, bottom);

            if(valve_signal & (1L << v))
                my_fill_rect(&signal_rect, 0x00ff0000, skip);
            else
                my_fill_rect(&signal_rect, 0x00000000, skip);
        }
}
//----------------------------------------------------------------------------------
// ----- display travel speed -----

void show_speed(float speed)    // speed is in ravens/tick
{
    char string[10] = {0};
    static Rect bar_rect;

    // do bar indicator
        my_fill_rect(&bar_rect, INSET_BG, 1);

        bar_rect.left = speed_rect.left + speed*10;
        bar_rect.right = bar_rect.left + 5;

        bar_rect.top = speed_rect.top+1;
        bar_rect.bottom = speed_rect.bottom-1;

        if(bar_rect.right >= speed_rect.right)
        {
            bar_rect.right = speed_rect.right;
            bar_rect.left = bar_rect.right - 5;
            my_fill_rect(&bar_rect, 0x00ffff00, 1);
        }
        else
            my_fill_rect(&bar_rect, 0x00000000, 1);

        my_fill_rect(&speed_disp_rect, WHITE, 1);

    // display speed
        //sprintf(string, "%3.1f m/s", (speed * 0.462);    // display m/s
        sprintf(string, "%3.1f mph", speed*1.032);       // display mph
    do_text_rect(speed_disp_rect, string, 2);

}
//-----------------------------------------------------------------------------------------
// ----- initialize video digitizer (see ROps_control.h) -----

short vd_initialize(video_digitizer *vd)
{
    short err=0;
    int max_width, max_height;
    Rect *vd_maxrect;

    myprint(status_line, "Opening video digitizer driver", 0);
    if(err=openVideo(&(*vd).refnum, &(*vd).base))
        return(err);
    myprint(status_line, "Selecting video digitizer", 0);
    if(err=vd_select_board(&(*vd).params, (*vd).refnum, 1))        // select first (1) vd
        return(err);
    myprint(status_line, "Resetting video digitizer", 0);
    if(err=vd_reset(&(*vd).params))
        return(err);
    myprint(status_line, "Setting video digitizer source type", 0);
    (*vd).source = COMPOSITE;        // 0 = COMPOSITE, 1 = S_VIDEO
    if(err=vd_set_source(&(*vd).params, (*vd).source))
        return(err);
    myprint(status_line, "Setting video digitizer speed", 0);
    if(err=vd_set_speed(&(*vd).params))
        return(err);

    myprint(status_line, "Finding video digitizer max. size", 0);
    if(err=vd_find_max_rect(&(*vd).params, &max_width, &max_height))
        return(err);

    myprint(status_line, "Finding video digitizer max. size", 0);
    if(err=vd_find_max_source_rect(&(*vd).params, vd_maxrect))
        return(err);

        (*vd).image_rect.top = (*vd_maxrect).top + 4;
        (*vd).image_rect.left = (*vd_maxrect).left + 2;
        (*vd).image_rect.bottom = ((*vd).image_rect.top + max_height - 14);
        (*vd).image_rect.right = (*vd).image_rect.left + max_width - 10;

        (*vd).disp_rect.top = (*vd).image_rect.top + VD_1_WINDOW_TOP - 8;
        (*vd).disp_rect.left = (*vd).image_rect.left + VD_1_WINDOW_LEFT - 4;
        (*vd).disp_rect.bottom = (*vd).image_rect.bottom + VD_1_WINDOW_TOP - 8;
        (*vd).disp_rect.right = (*vd).image_rect.right + VD_1_WINDOW_LEFT - 4;

        (*vd).width = (*vd).image_rect.right - (*vd).image_rect.left - 81;
        (*vd).height = (((*vd).image_rect.bottom - (*vd).image_rect.top));

        (*vd).start.v = VD_1_WINDOW_TOP;
        (*vd).start.h = VD_1_WINDOW_LEFT;

    myprint(status_line, "Setting video digitizer size", 0);
    if(err=vd_set_rect(&(*vd).params, &(*vd).image_rect, (*vd).width, (*vd).height, 1))
        return(err);
    myprint(status_line, "Setting video digitizer location", 0);
    if(err=vd_set_location(&(*vd).params, (*vd).start))
        return(err);
    myprint(status_line, "Setting video digitizer brightness and contrast", 0);
    if(err=vd_set_video_level(&(*vd).params, &(*vd).brightness, &(*vd).contrast))
        return(err);
    if(err=vd_set_hue_level(&(*vd).params, &(*vd).hue));
        return(err);
    if(err=vd_set_sat_level(&(*vd).params, &(*vd).saturation));
        return(err);

    if(err = vd_set_nubus(&(*vd).params, 255))
        return(err);

    return(0);

}
//-----------------------------------------------------------------------------------------
// ----- set up display window -----

short set_video_display(video_digitizer *vd, short width, short height, Point start, short mode)
{
    short err;

    if(err=vd_set_rect(&(*vd).params, &(*vd).image_rect, width, height, mode))
        return(err);

    if(err=vd_set_location(&(*vd).params, start))
        return(err);

    return(0);
}
//-----------------------------------------------------------------------------------------
// ----- read valve information from configuratin file -----

void get_valve_data(valves valve[], short *valve_count)
{
    char string[200] = {0};
    FILE *in_file;
    short vnumber;

    myprint(status_line, "loading valve data", 0);
    if(in_file = fopen("taos.ini","r"))
    {
        while(strncmp(string, "VALVE", 5))    // read through file line by line until
            fgets(string, 200, in_file);        // valve information title found
        fgets(string, 200, in_file);            // read valve data heading

        vnumber = 0;
        while(1)                                // read valve data until END
        {
            fgets(string, 200, in_file);
            if(!strncmp(string, "END", 3))
                break;
            valve[vnumber].set_valve(string);
            vnumber++;
        }
        *valve_count = vnumber;
        fclose(in_file);
    }
    else
        myprint(status_line, "Valve data file not read", 0);

    if(*valve_count > MAX_VALVE)
    {
        myprint(status_line, "Valve limit exceeded, check configuration file", 0);
        pause(5.0);
        exit(0);
    }
}
//-----------------------------------------------------------------------------------------
// ----- read the roi data in from configuration file -----

short get_roi_data(roi zone[MAX_SET+1][MAX_BAND+1], short *set_count, int control_increment,
   float band_adj[], short*band_count, unsigned long base)
{
    short    err=0;
    int      band = 0;
    float    set_width;
    FILE     *in_file;
    char string[200] = {0};
    float roi_voffsetm;
    short vd_number = 0;

    // find width of set in pixels (nearest band)
       set_width = (float)control_increment / RAVENS;

    // find meters to middle of image
       roi_voffsetm = ((float)(vd[vd_number].disp_rect.bottom
          - vd[vd_number].disp_rect.top)/2.0) / (vd[vd_number].v_coef0);

    // read data from file
    if(in_file = fopen("taos.ini","r"))
    {
       // find start of roi data in file
       while(strncmp(string, "ROI", 3))
          fgets(string, 100, in_file);
       fgets(string, 100, in_file);

       // read leading roi data for each band in file
       while(1)
       {
          fgets(string, 200, in_file);
          if(!strncmp(string, "END", 3))
             break;
          zone[0][band].set_roi(string, set_width, &band_adj[band], base, roi_voffsetm);
          band++;
       }

       // record number of bands read
       *band_count = band;

       if(*band_count > MAX_BAND)
       {
          myprint(status_line, "Band limit exceeded, maximum: ", MAX_BAND);
          pause(5.0);
          exit(0);
       }

       // close file and proceed
       fclose(in_file);
       myprint(status_line,"\nregion information read\n",0);
    }
    else
       myprint(status_line,"\nregion information error\n",err);

    // define remaining regions
    make_rois(zone, band_count, set_count, base);

    return(err);
}
//------------------------------------------------------------------------------
// ----- define additional roi's in based on first set -----

short make_rois(roi zone[MAX_SET+1][MAX_BAND+1], short *band_count, short *set_count,
   unsigned long base)
{
    short largest_v, band_largest_v, set, shift, band;

  // determine how many control sets fit in image
       largest_v = zone[0][0].find_max_vpixel();
       *set_count = max_image_v / largest_v;
       largest_v *= *set_count;

       if(*set_count > MAX_SET)
       {
          myprint(status_line, "Set limit exceeded, maximum: ", MAX_SET);
          pause(5.0);
          exit(0);
       }

    // add more roi's to each band
       for(band = 0; band < *band_count; band++)
       {
          for(set = 1; set < *set_count; set++)
          {
             zone[set][band].add_roi(&zone[set-1][band]);
          }
       }

    // adjust roi's into video window
       for(band = 0; band < *band_count; band++)
       {
          band_largest_v = zone[*set_count - 1][band].find_max_vpixel();
          shift = (largest_v - band_largest_v)/2;
          for(set = 0; set < *set_count; set++)
          {
             zone[set][band].adjust_roi(shift, base);
          }
       }

    return(0);
}
// ------------------------------------------------------------------------------
// ----- allow adjustment of video digitizer board (image quality) -----

void do_image_adjust(short vd_number, short *brightness, short *contrast, short *hue,
   short *saturation)
{
   short err = 0;
   Point mouseloc;
   short split_flag=0, height, width, lut_flag = 0;
   Point start;
   short first_vd_right;
   roi   big_zone;

   // initial settings
      first_vd_right = vd[vd_number].image_rect.right;
      vd[vd_number].image_rect.right = vd[vd_number].disp_rect.right;
      width = vd[vd_number].image_rect.right - vd[vd_number].image_rect.left;
      height = vd[vd_number].image_rect.bottom - vd[vd_number].image_rect.top;
      vd_set_rect(&vd[vd_number].params, &vd[vd_number].image_rect, width, height, 1);

   // define full image zone for CLUT application
      big_zone.make_big_zone(&vd[vd_number].disp_rect);

   // find locations for even / odd display windows
      height = vd[0].height/2 - 1;
      start.h = vd[0].start.h;
      start.v = vd[0].start.v + height;

   // draw video digitizer adjustment window and controls
      grab_frame(1, vd[vd_number].refnum, WAIT);
      status.run = OFF;
      draw_main_screen();
      draw_vd_adjust_area(ON);
      my_draw_v_level(vd_contrast_rect, *contrast, 0, 255);
      my_draw_v_level(vd_bright_rect, *brightness, 0, 255);
    my_draw_v_level(vd_hue_rect, *hue, 0, 255);
    my_draw_v_level(vd_sat_rect, *saturation, 0, 255);
    if(vd[vd_number].source == S_VIDEO)
    {
      my_fill_rect(&s_video_rect, BLACK, 1);
      my_fill_rect( &composite_rect, WHITE, 1);
    }
    else
    {
      my_fill_rect(&s_video_rect, WHITE, 1);
      my_fill_rect(&composite_rect, BLACK, 1);
    }

    my_fill_rect(&split_video_rect, WHITE, 1);
    my_fill_rect(&interlaced_video_rect, BLACK, 1);

    cont_frame(&vd[0].params, DONTWAIT);

  // do adjustments
  while(!mouse_in(vd_adj_finished_rect, mouseloc))
  {
    WaitNextEvent(everyEvent, &gEvent, LONG_NAP, NO_CURSOR);
      if(gEvent.what == mouseDown)
      {
        GetMouse(&mouseloc);
        do
        {
          if(mouse_in(vd_more_contrast_rect, mouseloc))
            *contrast += 1;
          else if(mouse_in(vd_less_contrast_rect, mouseloc))
            *contrast -= 1;
          else if(mouse_in(vd_contrast_rect, mouseloc))
            *contrast = 255 * (vd_contrast_rect.bottom - mouseloc.v)
              / (vd_contrast_rect.bottom - vd_contrast_rect.top);
          else if(mouse_in(vd_more_bright_rect, mouseloc))
            *brightness += 1;
          else if(mouse_in(vd_less_bright_rect, mouseloc))
            *brightness -= 1;
          else if(mouse_in(vd_bright_rect, mouseloc))
            *brightness = 255 * (vd_bright_rect.bottom - mouseloc.v)
              / (vd_bright_rect.bottom - vd_bright_rect.top);
          else if(mouse_in(vd_hue_rect, mouseloc))
            *hue = 255 * (vd_hue_rect.bottom - mouseloc.v)
              / (vd_hue_rect.bottom - vd_hue_rect.top);
          else if(mouse_in(vd_sat_rect, mouseloc))
            *saturation = 255 * (vd_sat_rect.bottom - mouseloc.v)
              / (vd_sat_rect.bottom - vd_sat_rect.top);
          else if(mouse_in(s_video_rect, mouseloc))       // set for s-video input
          {
            HideCursor();
            vd[vd_number].source = S_VIDEO;
            my_fill_rect(&s_video_rect, BLACK, 1);
            my_fill_rect(&composite_rect, WHITE, 1);
            ShowCursor();
          }
          else if(mouse_in(composite_rect, mouseloc))   // set for composite input
          {
            HideCursor();
            vd[vd_number].source = COMPOSITE;
            my_fill_rect(&s_video_rect, WHITE, 1);
            my_fill_rect(&composite_rect, BLACK, 1);
            ShowCursor();
          }
          else if(mouse_in(split_video_rect, mouseloc)) // display odd/even separately
          {
            HideCursor();
            split_flag = 1;
            my_fill_rect(&split_video_rect, BLACK, 1);
            my_fill_rect(&interlaced_video_rect, WHITE, 1);
            ShowCursor();
          }
          else if(mouse_in(interlaced_video_rect, mouseloc)) // normal display
          {
            HideCursor();
            split_flag = 0;
            my_fill_rect(&split_video_rect, WHITE, 1);
            my_fill_rect(&interlaced_video_rect, BLACK, 1);
            set_video_display(&vd[0], width, vd[0].height, vd[0].start, 1);
            cont_frame(&vd[0].params, DONTWAIT);
            ShowCursor();
          }
          else if(mouse_in(apply_lut_rect, mouseloc))     // apply the CLUT to image
          {
            HideCursor();
            lut_flag = (lut_flag == 0) ? 1 : 0;
            if(lut_flag)
              my_fill_rect(&apply_lut_rect, BLACK, 1);
            else
            {
              my_fill_rect(&apply_lut_rect, WHITE, 1);
              cont_frame(&vd[0].params, DONTWAIT);
            }
            ShowCursor();
          }

          // update values
            vd[vd_number].brightness = *brightness;
            vd[vd_number].contrast = *contrast;
            vd[vd_number].hue = *hue;
            vd[vd_number].saturation = *saturation;

          // seperate odd / even display
            if(split_flag)
            {
              set_video_display(&vd[0], width, height, vd[0].start, 1);
              grab_frame(1, vd[0].refnum, WAIT);
              set_video_display(&vd[0], width, height, start, 2);
              grab_frame(1, vd[0].refnum, WAIT);
            }

          // do adjustments to digitizer board

          if(err=vd_set_video_level(&vd[vd_number].params, &vd[vd_number].brightness,
            &vd[vd_number].contrast))
              myprint(status_line, "video digitizer error: ", err);

          if(err=vd_set_hue_level(&vd[vd_number].params, &vd[vd_number].hue))
              myprint(status_line, "video digitizer error: ", err);

          if(err=vd_set_sat_level(&vd[vd_number].params, &vd[vd_number].saturation))
              myprint(status_line, "video digitizer error: ", err);

          if(err=vd_set_source(&vd[vd_number].params, vd[vd_number].source))
            myprint(status_line, "video digitizer error: ", err);

          // update control bar levels in display
            my_draw_v_level(vd_contrast_rect, *contrast, 0, 255);
            my_draw_v_level(vd_bright_rect, *brightness, 0, 255);
            my_draw_v_level(vd_hue_rect, *hue, 0, 255);
            my_draw_v_level(vd_sat_rect, *saturation, 0, 255);

        } while(StillDown()); // stay in loop if mouse is held down
        }

        // do odd / even display
        if(split_flag)
        {
           set_video_display(&vd[0], width, height, vd[0].start, 1);
           grab_frame(1, vd[0].refnum, WAIT);
           set_video_display(&vd[0], width, height, start, 2);
           grab_frame(1, vd[0].refnum, WAIT);
        }

        // show lut results
        if(lut_flag)
        {
           grab_frame(1, vd[0].refnum, WAIT);
           big_zone.analyze_roi(0, 1);
        }
    }

    // done - reset things and clean up
      vd[0].start.v = VD_1_WINDOW_TOP;
      vd[0].start.h = VD_1_WINDOW_LEFT;
      vd[0].image_rect.right = zone[0][band_count-1].find_max_hpixel() + 8;
      vd[0].width = vd[0].image_rect.right - vd[0].image_rect.left;

      set_video_display(&vd[0], vd[0].width, vd[0].height, vd[0].start, 1);

      draw_vd_adjust_area(OFF);

      draw_main_screen();
      draw_video_area(ON);
      draw_data_area(ON);
      draw_over_speed_area(ON);
      draw_speed_area(ON);
      draw_controls(ON);

      grab_frame(1, vd[vd_number].refnum, WAIT);
}
// --------------------------------------------------------------------------------
// ----- pattern control -----
void do_pattern_control(unsigned char mode)
{
      HideCursor();
      if (mode)
      {
         status.expand = OFF;
         my_fill_rect(&pattern_control_rect, WHITE, 1);
      }
      else
      {
         status.expand = ON;
         my_fill_rect(&pattern_control_rect, BLACK, 1);
      }
      ShowCursor();
}

// --------------------------------------------------------------------------------
// ----- run mode control -----
void do_run_mode_control(unsigned char mode)
{
      HideCursor();
      if (!mode)
      {
         stop_running();
         my_fill_rect(&mode_control_rect, WHITE, 1);
      }
      else
      {
         start_running();
         my_fill_rect(&mode_control_rect, BLACK, 1);
      }
      ShowCursor();

}
// --------------------------------------------------------------------------------
// ----- shadow control -----
void do_shadow_control(unsigned char mode)
{
      HideCursor();
      if (mode)
      {
         status.shadows = OFF;
         my_fill_rect(&shadow_control_rect, WHITE, 1);
         lutp = lutp1;
      }
      else
      {
         status.shadows = ON;
         my_fill_rect(&shadow_control_rect, BLACK, 1);
         lutp = lutp2;
      }
      ShowCursor();
}
// --------------------------------------------------------------------------------
// ----- modify the roi definitions -----

void do_roi_adjust()
{
   short err = 0;
   Point mouseloc;
   int i, ivalue;
   Rect rect;
   float fvalue;
   char string[50] = {0};
   short dir;
   short   set, shift;
   short band;
   float roi_voffsetm;

   short vd_number = 0;

   // set up roi adjustment window
      status.run = OFF;
      draw_main_screen();
      draw_roi_adjust_area(ON, band_count, zone);

   // remove adjustment of roi's into video window
      for(band = 0; band < band_count; band++)
      {
         shift = zone[0][band].find_min_vpixel();
         for(set = 0; set < set_count; set++)
            zone[set][band].reset_roi(shift, vd[0].base);
      }

   // find mouse click and proceed with edits
      while(!mouse_in(roi_finished_rect, mouseloc))
      {
         WaitNextEvent(everyEvent, &gEvent, LONG_NAP, NO_CURSOR);
         if(gEvent.what == mouseDown)
         {
            GetMouse(&mouseloc);
            for(i=0; i<band_count; i++)
        {
           if(mouse_in(band_up[i], mouseloc))
           {
              dir = 1;
              break;
           }
           else if(mouse_in(band_down[i], mouseloc))
           {
              dir = -1;
              break;
           }
        }

     if(i < band_count)
     {
        if(mouse_in(near_side_rect, mouseloc))
        {
           fvalue = zone[0][i].roi_nearside();
           do
           {
              fvalue += 0.01 * dir;
              if(fvalue < 0.0)
                 fvalue = 0.0;
              sprintf(string, "%3.2f", fvalue);
              update_roi_display(&near_side_rect, &band_rect[i], string);
              zone[0][i].edit_roi_nearside(fvalue, vd[0].base, &band_offset_adj[i]);
              pause(0.1);
           }while(StillDown());
        }

        if(mouse_in(far_side_rect, mouseloc))
        {
           fvalue = zone[0][i].roi_farside();
           do
           {
              fvalue += 0.01 * dir;
              if(fvalue < 0.0)
                 fvalue = 0.0;
              sprintf(string, "%3.2f", fvalue);
              update_roi_display(&far_side_rect, &band_rect[i], string);
              zone[0][i].edit_roi_farside(fvalue);
              pause(0.1);
           }while(StillDown());
        }

        if(mouse_in(horz_rect, mouseloc))
        {
           ivalue = zone[0][i].roi_horz_skip();
           do
           {
              ivalue += 1 * dir;
              if(ivalue < 1)
                 ivalue = 1;
              sprintf(string, "%i", ivalue);
              update_roi_display(&horz_rect, &band_rect[i], string);
              zone[0][i].edit_roi_h_skip(ivalue);
              pause(0.1);
           }while(StillDown());
        }

        if(mouse_in(vert_rect, mouseloc))
        {
           ivalue = zone[0][i].roi_vert_skip();
           do
           {
              ivalue += 1 * dir;
              if(ivalue < 1)
                 ivalue = 1;
              sprintf(string, "%i", ivalue);
              update_roi_display(&vert_rect, &band_rect[i], string);
              zone[0][i].edit_roi_v_skip(ivalue);
              pause(0.1);
           }while(StillDown());
        }

        if(mouse_in(noise_rect, mouseloc))
        {
           ivalue = zone[0][i].roi_noise();
           do
           {
              ivalue += 1 * dir;
              if(ivalue < 0)
                 ivalue = 0;
              sprintf(string, "%i", ivalue);
              update_roi_display(&noise_rect, &band_rect[i], string);
              zone[0][i].edit_roi_noise(ivalue);
              pause(0.1);
           }while(StillDown());
        }
     }

     if(mouse_in(more_control_width_rect, mouseloc) || mouse_in(less_control_width_rect,
        mouseloc))
     {
        do
        {
           if(mouse_in(more_control_width_rect, mouseloc))
              valve_action_increment++;
           else
              valve_action_increment--;
           if(valve_action_increment < 1)
              valve_action_increment = 1;
           sprintf(string, " Control Width: %3.2f m", (float)valve_action_increment
              / RAVENS);
           my_fill_rect(&control_width_rect, WHITE, 1);
           my_draw_rect(control_width_rect, BLACK);
           do_text_rect(control_width_rect, string, 0);
           pause(0.1);
        }while(StillDown());
        for(i=0; i< band_count; i++)
           zone[0][i].edit_control_width(valve_action_increment);
     }

     if(mouse_in(more_image_rect,  mouseloc) || mouse_in(less_image_rect, mouseloc))
     {
        do
        {
           my_fill_rect(&less_image_rect, 0x00ff0000, 2);
           my_fill_rect(&more_image_rect, 0x0000ff00, 2);

           if(mouse_in(less_image_rect, mouseloc))
              max_image_v--;
           else if(mouse_in(more_image_rect, mouseloc))
              max_image_v++;
           sprintf(string, " Image Width: %i pixels", max_image_v);
           my_fill_rect(&image_width_rect, WHITE, 1);
           my_draw_rect(image_width_rect, BLACK);
           do_text_rect(image_width_rect, string, 0);
           pause(0.01);
        }while(StillDown());
        max_offset = VD_1_WINDOW_TOP + vd[0].image_rect.bottom - 4 - max_image_v;
     }
```

```
        if(mouse_in(add_roi_rect, mouseloc))
        {
          band_count++;
          sprintf(string, "%i %f %f %i %i %i", band_count-1,
             zone[0][band_count-2].roi_farside(), zone[0][band_count-2].roi_farside(),
             zone[0][band_count-2].roi_horz_skip(), zone[0][band_count-2].roi_vert_skip(),
             zone[0][band_count-2].roi_noise());

          roi_voffsetm = (float)((vd[vd_number].disp_rect.bottom
             - vd[vd_number].disp_rect.top)/2.0) / (vd[vd_number].v_coef0);

          zone[0][band_count-1].set_roi(string, (float)valve_action_increment / RAVENS,
             &band_offset_adj[band_count-1], vd[0].base, roi_voffsetm);

          shift = zone[0][band_count-1].find_min_vpixel();
          zone[0][band_count-1].reset_roi(shift, vd[0].base);

          rect.top = band_rect[band_count-2].top;
          rect.left = band_rect[band_count-2].left - 5;
          rect.right = band_rect[band_count-2].right + 5;
          rect.bottom = delete_roi_rect.bottom + 5;

          initialize_display(band_count, valve_count);

          HideCursor();
          my_fill_rect(&rect, MAIN_BG, 1);
          draw_roi_adjust_area(ON, band_count, zone);
          ShowCursor();
        }

        if(mouse_in(delete_roi_rect, mouseloc))
        {
          band_count--;
          rect.top = band_rect[band_count].top;
          rect.left = band_rect[band_count].left - 5;
          rect.right = band_rect[band_count].right + 5;
          rect.bottom = delete_roi_rect.bottom + 5;

          HideCursor();
          my_fill_rect(&rect, MAIN_BG, 1);
          draw_roi_adjust_area(ON, band_count, zone);
          ShowCursor();
        }
      }
    }

    vd[0].image_rect.right = zone[0][band_count-1].find_max_hpixel() + 8;
    vd[0].width = vd[0].image_rect.right - vd[0].image_rect.left;

    if(err=vd_set_rect(&vd[0].params, &vd[0].image_rect, vd[0].width, vd[0].height, 1))
    {
      myprint(status_line, "vd error: ", err);
      pause(5.0);
      exit( err);
    }


    // find meters to middle of image
      roi_voffsetm = ((float)(vd[vd_number].disp_rect.bottom - vd[vd_number].disp_rect.top)
        /2.0) / (vd[vd_number].v_coef0);

    // shift first zone in each band
      for(band = 0; band < band_count; band++)
        zone[0][band].shift_band(roi_voffsetm);
```

```
  make_rois(zone, &band_count, &set_count, vd[0].base);


  max_offset =(vd[0].disp_rect.bottom - vd[0].disp_rect.top) - (set_count) * (zone[0][0].find_max_vpixel() -
zone[0][0].find_min_vpixel());

  if(max_offset < 0)
     max_offset = 0;

  draw_main_screen();
  draw_video_area(ON);
  draw_data_area(ON);
  draw_over_speed_area(ON);
  draw_speed_area(ON);
  draw_controls(ON);
}
// ---------------------------------------------------------------------------------
// ----- change table values -----

void update_roi_display(Rect *v_rect, Rect *h_rect, char *string)
{
      Rect rect;

      SetRect(&rect, (*v_rect).left, (*h_rect).top, (*v_rect).right-10, (*h_rect).bottom);
      my_fill_rect(&rect, WHITE, 1);
      my_draw_rect(rect, BLACK);
      SetRect(&rect, (*v_rect).left, (*h_rect).top, (*v_rect).right, (*h_rect).bottom);
      do_text_rect(rect, string, 1);
}
// ---------------------------------------------------------------------------------
// ----- adjust valve coefficients -----

void do_valve_adjust()
{

      short i, dir, ivalue;
      float fvalue;
      Point mouseloc;
      char string[50] = {0};

      // set up for editing valve data
         status.run = OFF;
         draw_main_screen();
         draw_valve_adjust_area(ON, valve_count, valve);

      // modify settings
      while(!mouse_in(vedit_finished_rect, mouseloc))
      {
         WaitNextEvent(everyEvent, &gEvent, LONG_NAP, NO_CURSOR);
         if(gEvent.what == mouseDown)
         {
           GetMouse(&mouseloc);
           for(i=0; i< valve_count; i++)
           {
             if(mouse_in(vedit_up[i], mouseloc))
             {
               dir = 1;
               break;
             }
             else if(mouse_in(vedit_down[i], mouseloc))
             {
               dir = -1;
               break;
             }
           }
if(i < valve_count)
{
  // adjust distance from camera to valve
  if(mouse_in(dist_rect, mouseloc))
  {
    fvalue = valve[i].get_dist();
    do
    {
      fvalue += 0.01 * dir;
      if(fvalue < 0.0)
        fvalue = 0.0;
      sprintf(string, "%4.3f", fvalue);
      update_roi_display(&dist_rect, &vedit_rect[i], string);
      valve[i].edit_dist(fvalue);
      pause(0.1);
    }while(StillDown());
  }

  // adjust time of flight (tof) adjustment
  if(mouse_in(tof_rect, mouseloc))
  {
    fvalue = valve[i].get_tof();
    do
    {
      fvalue += 0.005 * dir;
      if(fvalue < 0.0)
        fvalue = 0.0;
      sprintf(string, "%4.3f", fvalue);
      update_roi_display(&tof_rect, &vedit_rect[i], string);
      valve[i].edit_tof(fvalue);
      pause(0.1);
    }while(StillDown());
  }

  // adjust f_lead
  if(mouse_in(f_lead_rect, mouseloc))
  {
    fvalue = valve[i].get_f_lead();
    do
    {
      fvalue += 0.005 * dir;
      if(fvalue < 0.0)
        fvalue = 0.0;
      sprintf(string, "%4.3f", fvalue);
      update_roi_display(&f_lead_rect, &vedit_rect[i], string);
      valve[i].edit_f_lead(fvalue);
      pause(0.1);
    }while(StillDown());
  }

  // adjust f_lag
  if(mouse_in(f_lag_rect, mouseloc))
  {
    fvalue = valve[i].get_f_lag();
    do
    {
      fvalue += 0.005 * dir;
      if(fvalue < 0.0)
        fvalue = 0.0;
      sprintf(string, "%4.3f", fvalue);
      update_roi_display(&f_lag_rect, &vedit_rect[i], string);
      valve[i].edit_f_lag(fvalue);
      pause(0.1);
    }while(StillDown());
  }

  // adjust d_lead
  if(mouse_in(d_lead_rect, mouseloc))
  {
    fvalue = valve[i].get_d_lead();
    do
    {
      fvalue += 0.005 * dir;
      if(fvalue < 0.0)
        fvalue = 0.0;
      sprintf(string, "%4.3f", fvalue);
      update_roi_display(&d_lead_rect, &vedit_rect[i], string);
      valve[i].edit_d_lead(fvalue);
      pause(0.1);
    }while(StillDown());
  }

  // adjust f_lag
  if(mouse_in(d_lag_rect, mouseloc))
  {
    fvalue = valve[i].get_d_lag();
    do
    {
      fvalue += 0.005 * dir;
      if(fvalue < 0.0)
        fvalue = 0.0;
      sprintf(string, "%4.3f", fvalue);
      update_roi_display(&d_lag_rect, &vedit_rect[i], string);
      valve[i].edit_d_lag(fvalue);
      pause(0.1);
    }while(StillDown());
  }

  // adjust band (inside)
  if(mouse_in(band_in_rect, mouseloc))
  {
    ivalue = valve[i].get_band_in();
    do
    {
      ivalue += 1 * dir;
      if(ivalue < 0)
        ivalue = 0;
      sprintf(string, "%i", ivalue);
      update_roi_display(&band_in_rect, &vedit_rect[i], string);
      valve[i].edit_band_in(ivalue);
      pause(0.1);
    }while(StillDown());
  }

  // adjust band (outside)
  if(mouse_in(band_out_rect, mouseloc))
  {
    ivalue = valve[i].get_band_out();
    do
    {
      ivalue += 1 * dir;
      if(ivalue < 0)
        ivalue = 0;
      sprintf(string, "%i", ivalue);
      update_roi_display(&band_out_rect, &vedit_rect[i], string);
      valve[i].edit_band_out(ivalue);
      pause(0.1);
    }while(StillDown());
  }
 }
}
}
```

```
   // done – reset the main display window
      draw_main_screen();
      draw_video_area(ON);
      draw_data_area(ON);
      draw_over_speed_area(ON);
      draw_speed_area(ON);
      draw_controls(ON);
}

// ------------------------------------------------------------------------------------
// ----- save configuration data ----

void save_settings(void)
{
   short outFileRefNum;
   unsigned char fileName[14] = {"\ptaos.ini"};

   StandardFileReply  outputReply;
   char string[256]= {0};
   short i;
   time_t systime;
   struct tm *currtime;

   systime = time(NULL);
   currtime = localtime(&systime);

   if(get_save_file(fileName, &outputReply))
   {
      if (!(FSpOpenDF (&(outputReply).sfFile, fsCurPerm, &outFileRefNum))) // Open data fork
      {
         gtheCursor = GetCursor(watchCursor);     // Change the cursor
         SetCursor(&**gtheCursor);

         // date
            sprintf(string, "%s", asctime(currtime));
            write_line(outFileRefNum, string, 0, 2);

         // control width
            write_line(outFileRefNum, "CONTROL WIDTH (m):", 0, 1);
            sprintf(string,"%4.2f",(float)valve_action_increment / RAVENS);
            write_line(outFileRefNum, string, 0, 1);
            write_line(outFileRefNum, "END CONTROL WIDTH", 0, 2);

         // image width
            write_line(outFileRefNum, "IMAGE WIDTH (p):", 0, 1);
            sprintf(string,"%hi", max_image_v);
            write_line(outFileRefNum, string, 0, 1);
            write_line(outFileRefNum, "END IMAGE WIDTH", 0, 2);

         // vd 1 image coefficients
            write_line(outFileRefNum, "VD1 IMAGE COEFFICIENTS:", 0, 1);
            sprintf(string, "%4.1f", vd[0].h_coef1);
            write_line(outFileRefNum, string, 0, 1);
            sprintf(string, "%4.1f", vd[0].h_coef2);
            write_line(outFileRefNum, string, 0, 1);
            sprintf(string, "%4.1f", vd[0].v_coef0);
            write_line(outFileRefNum, string, 0, 1);
            sprintf(string, "%4.1f", vd[0].v_coef1);
            write_line(outFileRefNum, string, 0, 1);
            sprintf(string, "%4.1f", vd[0].v_coef2);
            write_line(out FileRefNum, string, 0, 1);
            write_line(outFileRefNum, "END VD1 IMAGE COEFFICIENTS", 0, 2);

         // vd 1 settings
            write_line(outFileRefNum, "VD1 SETTINGS:", 0, 1);
            sprintf(string, "%i", vd[0].brightness);
            write_line(outFileRefNum, string, 0, 1);
            sprintf(string, "%i", vd[0].contrast);
            write_line(outFileRefNum, string, 0, 1);
            sprintf(string, "%i", vd[0].hue);
            write_line(outFileRefNum, string, 0, 1);
            sprintf(string, "%i", vd[0].saturation);
            write_line(outFileRefNum, st ring, 0, 1);
            write_line(outFileRefNum, "END VD1 SETTINGS", 0, 2);

         // roi data
            write_line(outFileRefNum, "ROI DATA (m):", 0, 1);
            write_line(outFileRefNum, "BAND   NEAR FAR    H_SKIP   V_SKIP   NOISE", 0, 1);
            for(i = 0; i<band_count; i++)
            {
               sprintf(string, " %i ", i);
               write_line(outFileRefNum, string, 1, 0);
               sprintf(string, "   %4.2f", zone[0][i].roi_nearside());
               write_line(outFileRefNum, string, 1, 0);
               sprintf(string, "  %4.2f", zone[0][i].roi_farside());
               write_line(outFileRefNum, string, 2, 0);
               sprintf(string, "%i", zone[0][i].roi_horz_skip());
               write_line(outFileRefNum, string, 2, 0);
               sprintf(string, "%i", zone[0][i].roi_vert_skip());
               write_line(outFileRefNum, string, 2, 0);
               sprintf(string, "%i", zone[0][i].roi_noise());
               write_line(outFileRefNum, string, 0, 1);
            }
            write_line(outFileRefNum, "END ROI DATA", 0, 2);

         // valve data
            write_line(outFileRefNum, "VALVE_CONTROL_DATA (m, m/(m/s), bands):", 0, 1);
            write_line(outFileRefNum, "VALVE   DIST    TOF ADJ      F LEAD     F LAG     D LEAD    D LAG
BAND IN   BAND OUT    EXP,N     EXP,T", 0, 1);
            for(i = 0; i < valve_count; i++)
            {
               sprintf(string, "%i ", i);
               write_line(outFileRefNum, string, 2, 0);
               sprintf(string, "%4.2f", valve[i].get_dist());
               write_line(outFileRefNum, string, 2, 0);
               sprintf(string, "%4.2f", valve[i].get_tof());
               write_line(outFileRefNum, string, 2, 0);
               sprintf(string, "%4.2f", valve[i].get_f_lead());
               write_line(outFileRefNum, string, 2, 0);
               sprintf(string, "%4.2f", valve[i].get_f_lag());
               write_line(outFileRefNum, string, 2, 0);
               sprintf(string, "%4.2f", valve[i].get_d_lead());
               write_line(outFileRefNum, string, 2, 0);
               sprintf(string, "%4.2f", valve[i].get_d_lag());
               write_line(outFileRefNum, string, 2, 0);
               sprintf(string, "%i", valve[i].get_band_in());
               write_line(outFileRefNum, string, 2, 0);
               sprintf(string, "%i", valve[i].get_band_out());
               write_line(outFile RefNum, string, 2, 0);
               sprintf(string, "%4.2f", valve[i].get_expand_norm());
               write_line(outFileRefNum, string, 2, 0);
               sprintf(string, "%4.2f", valve[i].get_expand_tang());
               write_line(outFileRefNum, string, 0, 1);
            }
            write_line(outFileRefNum, "END VALVE CONTROL DATA", 0, 2);
         }
      }

      SetCursor(&qd.arrow);              // Restore the cursor

   FSClose (outFileRefNum);
   FlushVol (NIL, (outputReply).sfFile.vRefNum);
}
// ------------------------------------------------------------------------------------------
```

```
/-------------------------------------------//
/    taos.h        /
//-------------------------------------------//

#define APPLE_MENU 128

#define WINDOW_TOP   40
#define WINDOW_LEFT  0

#define VD_1_WINDOW_TOP   46
#define VD_1_WINDOW_LEFT  10

struct control_status
{
   Boolean  run;
   Boolean  show_pixels;
   Boolean  show_data;
   Boolean  shadows;
   Boolean  expand;
   Boolean  distance_counter;
} status = {0, 0, 0, 0, 0, 1};

#define OVER_SPEED      0x01        // PC0 - overspeed = low
#define DISABLED        0x02        // PC1 - diabled = low , also on watchdog timeout
#define READY       0x04        // PC2 - ready = low, "INIT" led on if high
#define WATCHDOG          0x08        // PC3 - set when pulse low < 2 times / second
#define AUTO_MODE         0x10        // PC4 - not used
#define SPRAY_SHADOWS 0x20        // PC5 - low input = spray shadows
#define EXPAND_PATTERN  0x40          // PC6 - low input = expand pattern

enum {status_line}; //, info_line, error_line};
enum {NO, YES};
enum {OFF, ON};
enum {MANUAL_ON, MANUAL_OFF, AUTO};

#define RAVENS          130     // raven radar detector = 130 pulse/m
#define TICKS_PER_SEC   60          // macintosh ticks per second

#define PI   3.14159

#define LUTSIZE        256   // brightness/contrast lut size
#define DEFBRIGHTNESS   150    // default brightness
#define NEWCONTRAST     150    // default contrast
#define FBLEFT        0
```

```
#define FBTOP          20
#define VOFFSET      1152
#define DRHWXLTV        695     // driver hardware name
#define F_MIDV        243     // vertical center of digitized image

#define NIL 0

#define MAX_BAND 32
#define MAX_SET 32
#define MAX_VALVE 16
```

```
//-------------------------------------------//
/    taos_rsrc.h     //
//-------------------------------------------//

#include<Palettes.h>

#define LAST_MENU         5     // Number of menus
#define APPLE_MENU     128     // Apple menu id
#define CONTROL_MENU     129
#define IMAGE_MENU    130
#define VALVE_MENU    131
#define RUN_MODE_MENU   132

#define RESOURCE_ID      127     // Starting index into the menu array

#define ERROR_BOX_ID    130


// MENUS

// ----- ABOUT MENU -----------
#define ABOUT_BOX          1     // About box menu item # in Apple menu

// ----- CONTROL MENU ---------
#define RUN_NOW          1     // Open item # in CONTROL menu
//--------------------- 2
#define STOP_NOW         3     // Quit item # in file menu
#define QUIT_NOW         4

// ----- IMAGE MENU ------------
// LUT              1
#define SHOW_LUT         2
```

```
#define LOAD_LUT        3
//------------------- 4
// IMAGE               5
#define ADJUST_IMAGE    6
#define MAP_IMAGE       7
#define COPY_IMAGE      8

// ----- VALVE MENU -----------


// ----- RUN MODE MENU ---------
#define PIXEL_DISPLAY 1
#define DATA_DISPLAY   2


// RECTANGLES

#define RECT_IM_ADJ 130

// CONTROLS

#define ABOUT_BOX_ID       128        // Resource IDs for dialogs
#define IMAGE_MAP_BOX_ID  129

#define LONG_NAP      10L
#define NO_CURSOR      0L
#define APPEND_MENU      0
#define IN_FRONT    -1
#define CHAR_CODE_MASK  255

// a gworld for every window
typedef struct
{
   CWindowRecord wind;
   PicHandle      thePict;
   short          paletteCode;
   PaletteHandle  thePalette;
   GWorldPtr      windGWorld;
} WindNGWorld, *WindNGWorldPk;




//------------------------------------------//
/     ROps_control.h  /
//------------------------------------------//

#include "PIPDriver.h"
#include <Files.h>
#include <Devices.h>
#include <Slots.h>
#include <Types.h>
#include <Stdio.h>
#include <ROMDefs.h>
#include <stdio.h>
#include <fp.h>
//#include <math.h>

#define UTableBase  0x0000011C    // Base address of unit table
#define FIRSTSLOT     0           // slot manager info
#define LASTSLOT     16

#define   MAXCOLORVAL    255
```

```
#define   LUTSIZE         256
#define   MINCOLORVAL   0
#define   ORIGIN     127.5      // Half of 255, for contrast calculations
#define   APPLEPIE        3.1416015625 // Apple's fixed point value of pi

short vd_open_driver(short *refnum);
int openVideo(short *refnum, unsigned long *base);
short vd_select_board(struct CntrlParam *pb, short refnum, int vd_brd);
short vd_reset(struct CntrlParam *pb);
short fill_fb(struct  CntrlParam *pb, unsigned long color);
short vd_set_source(struct CntrlParam *pb, int source);
short vd_set_speed(struct CntrlParam *pb);
short vd_find_max_rect(struct CntrlParam *pb, int *max_width, int *max_height);
short vd_find_max_source_rect(struct CntrlParam *pb, Rect *vrect);
short vd_set_rect(struct CntrlParam *pb, Rect *vrect, short width, short height, short method);
short vd_set_location(struct CntrlParam *pb, Point topleft);
short vd_set_digitize_mode(struct CntrlParam *pb, int mode);
short vd_set_hue_level(CntrlParam *pb, short *hue);
short vd_set_video_level(CntrlParam *pb, short *brightness, short *contrast);
short vd_set_sat_level(CntrlParam *pb, short *sat);

short set_saturation(struct CntrlParam *pb, int saturation_step);
short vd_set_zoom(struct CntrlParam *pb, int zoom_level);
int grab_frame(short vd_number, short refnum, short mode);
short cont_frame(struct CntrlParam *pb, short mode);
int check_frame_grab(short refnum);
short vd_set_nubus(struct CntrlParam *pb, unsigned char delay);

void makebclut(short brightness, short contrast, unsigned char ramp[]);
//----------------------------------------------------------------------------------------
struct video_digitizer
   {
     CntrlParam    params;
     short         refnum;          // vd reference number
     Rect          image_rect;
     Rect          disp_rect;
     Point            start;
     short            width;        // scaled width
     short         height;        // scaled height
     unsigned long base;              // frame buffer base addresss
     unsigned long *lutp;           // lookup table pointer
     short            contrast;     // contrast setting
     short            brightness;     // bright setting
     short            hue;          // hue setting
     short            saturation;     // saturation setting
     short            source;       // source type
     float         h_coef1;         // first order horz.
     float         h_coef2;         // second order horz.
     float         v_coef0;         // zeroth order vert.
     float         v_coef1;         // first order vert.
     float         v_coef2;         // second order vert.
   };

video_digitizer vd[2];

GDevice  **pipgd;                // GDevice of display "carrying" PIP window

// open frame grabber driver
short vd_open_driver(short *refnum)
{
   short err = 0;
   if(err = OpenDriver("\p.RasterOps 24MxTV PIP", refnum))
      *refnum = 0;
   return(err);
}
//----------------------------------------------------------------------------------------
```

```
int openVideo(short *refnum, unsigned long *base)
{
  short       error;
  GDevice     **dev;
  SpBlock     spb;              // Slot Manager
  short       slot, notfound, slotfound;

  notfound = 1;
  for (slot=FIRSTSLOT; (slot<=LASTSLOT) && (notfound); slot++)  // Exit when first board is found
  {
    spb.spSlot = slot;
    spb.spID = 0;
    spb.spExtDev = 0;
    error = SNextSRsrc(&spb);       /* What's in the slot? */
    if ((!error) && (spb.spSlot == slot))
    {
      // Get Board Id number
      spb.spID = boardId;
      spb.spResult = 0;
      if (!(error = SReadWord(&spb)))
      {
        /* Compare the found board ID with familiar boards. */
        switch((short)spb.spResult)
        {
          case BOARD_ID_XLTV:
            error = OpenDriver("\p.RasterOps 24XLTV PIP", refnum);
            notfound=0;
            slotfound=slot;
            break;
          case BOARD_ID_MXTV:
            error = OpenDriver("\p.RasterOps 24MxTV PIP", refnum);
            notfound=0;
            slotfound=slot;
            break;
        }
      }
    }
  }
  *base = 0xf0000000 | ( slotfound << 24);

  if (error) return(error);
  if (notfound) return (notfound);

  dev=GetDeviceList();      /* Get the GDevice */
  while (dev)
  {
    if (slotfound == ((*((*(AuxDCE ****)UTableBase)[(-(*dev)->gdRefNum)-1]))->dCtlSlot)) pipgd=dev;
    dev=GetNextDevice(dev);
  }


  return(error);
}

//----------------------------------------------------------------------------------
// ----- select frame grabber card --------

short vd_select_board(struct CntrlParam *pb, short refnum, int vd_brd)
{
  short err = 0;
  (*pb).ioCRefNum = refnum;
  (*pb).ioCompletion = NIL;
  (*pb).ioVRefNum = 0;
  (*pb).csCode = CONTROL_SELECTPIPBOARD;
  (*pb).csParam[0] = vd_brd;                // video digitizer order
  err = PBControl((ParmBlkPtr)pb, false);
```

```
  return(err);
}
//----------------------------------------------------------------------------------
// reset the video digitizer
short vd_reset(struct CntrlParam *pb)
{
  short err = 0;
  (*pb).csCode = CONTROL_PIPRESET;
  err = PBControl((ParmBlkPtr)pb, false);
  return(err);
}
//----------------------------------------------------------------------------------
// set initial video digitizer RAM to black
short fill_fb(struct  CntrlParam *pb, unsigned long color)
{
  short err = 0;
  long          *fillvalue;
  fillvalue = (long *)&(*pb).csParam[0];
  *fillvalue = color;      // Set the grabber RAM to selected color

  (*pb).csCode = 9000; //CONTROL_SET_PIPERASEFRAMEBUFFER;
  err = PBControl((ParmBlkPtr)pb, false);
  return(err);
}
//----------------------------------------------------------------------------------
// set video digitizer source signal
short vd_set_source(struct CntrlParam *pb, int source)
{
  short err = 0;
  (*pb).csParam[0] = source;
  (*pb).csCode = CONTROL_SET_PIPVIDEOSOURCE;
  err = PBControl((ParmBlkPtr)pb, false);
  return(err);
}
//----------------------------------------------------------------------------------
// set NuBus delay
short vd_set_nubus(struct CntrlParam *pb, unsigned char delay)
{
  short err = 0;
  (*pb).csParam[0] = delay;
  (*pb).csCode = CONTROL_SET_PIPNUBUSDELAY;
  err = PBControl((ParmBlkPtr)pb, false);
  return(err);
}
//----------------------------------------------------------------------------------
// set framegrabber for full speed (30 frame/sec)
short vd_set_speed(struct CntrlParam *pb)
{
  short err=0;
  (*pb).csCode = CONTROL_SET_PIPSPEED;
  (*pb).csParam[0] = 0;
  err = PBControl((ParmBlkPtr)pb, false);
  return(err);
}
//----------------------------------------------------------------------------------
// find maximum video rectangle supported (max. width and height)
short vd_find_max_source_rect(struct CntrlParam *pb, Rect *vrect)
{
  short err=0;

  (*pb).csCode = STATUS_GET_PIPMAXRECT;
  if(err = PBStatus((ParmBlkPtr)pb, false))
    return(err);
  (*vrect).top = (*pb).csParam[0];          // top
  (*vrect).left = (*pb).csParam[1];         // left
  (*vrect).bottom = (*pb).csParam[2];       // bottom
```

```
   (*vrect).right = (*pb).csParam[3];      // right

   return(err);
 }
//-----------------------------------------------------------------------------------------------
short vd_find_max_rect(struct CntrlParam *pb, int *max_width, int *max_height)
{
   short err=0;

   (*pb).csCode = STATUS_GET_MAXDESTSIZE;
   err = PBStatus((ParmBlkPtr)pb, false);
   *max_height = (*pb).csParam[1];
   *max_width = (*pb).csParam[0];

   return(err);
}
//-----------------------------------------------------------------------------------------------
// set up digitized video image rectangle
   short vd_set_rect(struct CntrlParam *pb, Rect *vrect, short width, short height, short method)
   {
      short err = 0;

      (*pb).csCode = CONTROL_SET_PIPDIGITIZEVIDEORECT;
      (*pb).csParam[0] = (*vrect).top;         // top
      (*pb).csParam[1] = (*vrect).left;        // left
      (*pb).csParam[2] = (*vrect).bottom;      // bottom
      (*pb).csParam[3] = (*vrect).right;       // right
      (*pb).csParam[4] = width;                // width
      (*pb).csParam[5] = height;               // height
      (*pb).csParam[6] = method;                  // sampling method flag
      err = PBControl((ParmBlkPtr)pb, false);
      return(err);
   }
//-----------------------------------------------------------------------------------------------
// set screen location
   short vd_set_location(struct CntrlParam *pb, Point topleft)
   {
      short err=0;
      (*pb).csCode = CONTROL_SET_PIPSCREENPOSITION;   // Set location
      (*pb).csParam[0] = topleft.v;           // top of frame buffer on screen
      (*pb).csParam[1] = topleft.h;        // left of frame buffer on screen
      err = PBControl((ParmBlkPtr)pb, false);
      return(err);
   }
//-----------------------------------------------------------------------------------------------
// set up for one shot frame grab, no waiting
   short vd_set_digitize_mode(struct CntrlParam *pb, int mode)
   {
   short err=0;
   (*pb).csCode = CONTROL_PIPONESHOT;  // one shot video acquisition
   (*pb).csParam[0] = mode;
      err = PBControl((ParmBlkPtr)pb, false);
      return(err);
   }
//-----------------------------------------------------------------------------------------------
// hue adjustment
   short vd_set_hue_level(CntrlParam *pb, short *hue)
   {
      short err = 0;
      (*pb).csCode = CONTROL_SET_PIPHUE;
      (*pb).csParam[0] = *hue;
      err = PBControl((ParmBlkPtr)pb, false);
      return(err);
   }
//-----------------------------------------------------------------------------------------------
// saturation adjustment
```

```
   short vd_set_sat_level(CntrlParam *pb, short *sat)
   {
     short err = 0;
     (*pb).csCode = CONTROL_SET_PIPSATURATION;
     (*pb).csParam[0] = *sat;
     err = PBControl((ParmBlkPtr)pb, false);
     return(err);
   }


//-----------------------------------------------------------------------------------------------
// frame-grabber brightness and contrast lut control
   short vd_set_video_level(CntrlParam *pb, short *brightness, short *contrast)
   {
     short          err = 0;
     unsigned char  ramp[LUTSIZE];
     FILE           *fp;

     if(*brightness == 0 && *contrast == 0)
     {
        // read brightness and contrast settings from file
        if(fp = fopen("ct_frame_grabber.txt","r"))
        {
           fscanf(fp,"%i, %i", *brightness, *contrast);
           fclose(fp);
        }
        else
        {
           *brightness = DEFBRIGHTNESS;
           *contrast = NEWCONTRAST;
        }
     }

     // create look up table for frame-grabber control
        makebclut(*brightness, *contrast, ramp);

     // set frame-grabber contrast and brightness
        (*pb).csCode  = CONTROL_SET_DIGITIZELUT;  // 9044
        *((unsigned long *) &(*pb).csParam[0]) = (unsigned long) ramp;
        err = PBControl((ParmBlkPtr)pb, false);
        return(err);
   }


//-----------------------------------------------------------------------------------------------
// The C code below loads a 256-byte lookup table for a specific brightness and contrast.
// Supplied by RasterOps

/* Make brightness & contrast lookup table.
   Brightness is between 0 and 255; 0 is darkest, 255 is brightest.
   Contrast is between 0 and 255; 0 is minimum contrast; 255 is maximum
   contrast. If brightness=128 and contrast=128 the LUT is a linear ramp
   (default).
*/
//-----------------------------------------------------------------------------------------------
void makebclut( short brightness, short contrast, unsigned char *ramp)
{
   int      i;
   long        val;
   long        multfactor, originfactor;
   Fract       sine, cosine;
   Fixed    temp;
   Fixed       pi;
```

```
    long double applepi = APPLEPIE;
    long double origin = ORIGIN;

    if ((brightness == 128) && (contrast == 128))  /* Load linear ramp. */
    for (i = 0; i < LUTSIZE; i++)
       ramp[i] = i;
    else
    {
       brightness &= 0xFF;
       contrast &= 0xFF;
       temp = FixRatio(contrast/2, 256);
       pi = X2Fix(applepi);
       sine = FracSin(FixMul(pi,temp));
       cosine = FracCos(FixMul(pi, temp));
       multfactor = FixDiv(Frac2Fix(sine), Frac2Fix(cosine)); // tangent
       originfactor = FixMul(FixRatio(1,1) - multfactor, X2Fix(origin));

       for (i = 0; i < LUTSIZE; i++)
       {
          val = Fix2Long(FixMul(FixRatio(i,1), multfactor) + originfactor);
          val += (brightness - 128);
          if (val < MINCOLORVAL)
             val = MINCOLORVAL;
          else if (val > MAXCOLORVAL)
             val = MAXCOLORVAL;
          ramp[i] = val;
       }
    }
}

// ---------------------------------------------------------------------------

// set video digitizer saturation level
   short set_saturation(struct CntrlParam *pb, int saturation_step)
   {
      short err = 0;
      (*pb).csParam[0] += saturation_step;
      if((*pb).csParam[0] > 255 || (*pb).csParam[0] <0)
         SysBeep(1);
      (*pb).csCode = 9007;
      err = PBControl((ParmBlkPtr)pb, false);
      return(err);
   }
//---------------------------------------------------------------------------------
// set video digitizer hardware zoom
   short vd_set_zoom(struct CntrlParam *pb, int zoom_level)
   {
      short err = 0;
      (*pb).csParam[0] = zoom_level;
      (*pb).csCode = 9001;
      err = PBControl((ParmBlkPtr)pb, false);
      return(err);
   }

   //-------------------------------------------------------------------------
//---------------------------------------------------------------------------------
int grab_frame(short vd_number, short refnum, short mode)          //
{
   short int error, vert_shift = 0;
   static char top_image = 0;
   CntrlParam csblk;

   // select the frame grabber board
      csblk.ioCRefNum = refnum;
      csblk.ioCompletion = nil;
      csblk.ioVRefNum = 0;
      csblk.csCode = CONTROL_SELECTPIPBOARD;
      csblk.csParam[0] = vd_number;                   // SELECT video digitizer board
      if(error = PBControl((ParmBlkPtr)&csblk, false))
         return(error);

   // set run mode and trigger
      csblk.csCode = CONTROL_PIPONESHOT;  // one shot video acquisition
      csblk.csParam[0] = mode;        //WAIT; //2 = wait until halfdone
      error = PBControl((ParmBlkPtr)&csblk, false);

      return(error);
}

//-------------------------------------------------------------------------
short cont_frame(struct CntrlParam *pb, short mode)
{
   short err = 0;

   (*pb).csCode = CONTROL_PIPCONTINUOUS;  // one shot video acquisition
   (*pb).csParam[0] = 1;
   (*pb).csParam[1] = mode;
   err = PBControl((ParmBlkPtr)pb, false);

   return(err);

}
//-------------------------------------------------------------------------
int check_frame_grab(short refnum)          /
{
   CntrlParam pb;

   pb.ioCRefNum = refnum;
   pb.ioCompletion = nil;
   pb.ioVRefNum = 0;
   pb.csCode = STATUS_GET_PIPACTIVE;
   PBStatus((ParmBlkPtr)&pb, false);

   return(pb.csParam[0]);       // 1=active, 0=not active
}

//-------------------------------------------------------------------------





/-----------------------------------------//
/    taos_utility.h    /
//----------------------------------------//

// ---------------------------------------------------------------------------
/
/ taos_utility.h
/
/ September 1994
/
/ Chris Tauzer

/ This file is used for the utility functions in support of the MVS control
/ program.
```

```
//------------------------------------------------------------------------------

#include <string.h>
#include <StandardFile.h>

#define LEFT_FR      0x00cccccc
#define TOP_FR       0x00444444
#define RIGHT_FR     0x00444444
#define BOTTOM_FR    0x00cccccc

#define MAIN_BG      0x00777777
#define INSET_BG     0x00999999
#define WHITE        0x00ffffff
#define BLACK        0x00000000
#define RED          0x00ff0000
#define GREEN        0x0000ff00
#define BLUE         0x000000ff
#define DKGRAY       0x00333333
#define MEDGRAY      0x00888888
#define LTGRAY       0x00bbbbbb

#define ONE_FILE_TYPE 1
#define NEW_LINE     0x0D
#define TAB          0x09

//------------------------------------------------------------------------------

Rect status_rect, info_rect, error_rect;

struct circle
{
   Point   center;
   short   radius;
};

OSType  gfileCreator = {'MPCC'};
OSType  gfileType = {'TEXT'};
CursHandle gtheCursor;

//------------------------------------------------------------------------------

void pause(float pause_time);
void color_screen(unsigned long color);
void my_fill_rect(Rect *new_rect, unsigned long color, char skip);
void my_draw_point(short x, short y, unsigned long color);
void my_draw_frame(Rect *fr, char fw);
void my_draw_frame_in(Rect *fr, char fw);
void myprint(char info_type, char *message, int value);
float my_round(float number);
void my_draw_rect(Rect rect, unsigned long color);
//void Ask_File(void);
short get_save_file(unsigned char fileName[], StandardFileReply *outputReply);
void report_error(OSErr errorCode);
void write_line(short outFileRefNum, char *message, short tabs, short lines);
void write_control(short outFileRefNum, unsigned char control);
void write_data(short outFileRefNum, char *string);
void new_line(short outFileRefNum, short line_count);
void tab(short outFileRefNum, short tab_count);
void my_draw_circle(circle *circle, unsigned long color);
void my_fill_circle(circle *circle, unsigned long color);
void my_draw_line(Point p1, Point p2, unsigned long color);
void my_draw_v_level(Rect level_rect, short level, short min, short max);
void my_draw_h_bilevel(Rect level_rect, short level, short min, short max);
void my_draw_v_bilevel(Rect level_rect, short level, short min, short max);
void gaussian_solve_3(double in[], double out[]);

//------------------------------------------------------------------------------
// pause for "pause_time" seconds
void pause(float pause_time)
{
   int start = TickCount();

   while(TickCount() - start < pause_time*60.0);

}  //end of pause()

//------------------------------------------------------------------------------

void color_screen(unsigned long color)
{
   int h, v, hstep, vstep;
   int maxh, maxv, minh, minv;

   unsigned long hpxl_step, vpxl_step;
   unsigned long *pxl_color;
   unsigned long pxl_addr, base;


     base = vd[0].base;

     hstep = 1;                    // horz pixels skipped
     vstep = 1;                    // vert pixels skipped
     minh = 20;
     maxh = 812;                         // horz screen pixels
     minv = 20;
     maxv = 608;                         // vert screen pixels

     hpxl_step = hstep * 0x00000004;      // horz memory increment
     vpxl_step = 0x00000d00;              // vert memory increment

     pxl_addr = base + (minv * vpxl_step + minh * hpxl_step);  // first pixel

     for(v=minv; v<maxv; v+=vstep)          // select row
     {
        for(h=minh; h<maxh; h+=hstep)       // select column
        {
           pxl_color = (unsigned long *) pxl_addr;     // get pixel location
           *pxl_color =  color;                // change pixel color
           pxl_addr += (unsigned long)hpxl_step;    //  move to next pixel column
        }

        pxl_addr = base + (v * vpxl_step + minh * hpxl_step);    // next pixel row

     }

}

//------------------------------------------------------------------------------
void my_fill_rect(Rect *new_rect, unsigned long color, char skip)
{
   int h, v;
   unsigned long hpxl_step, vpxl_step;
   unsigned long *pxl_color;
   unsigned long pxl_addr, base;

   base = vd[0].base;

   hpxl_step = 0x00000004;      // horz memory increment
   vpxl_step = 0x00000d00;      // vert memory increment
```

```
    pxl_addr = base + (( *new_rect).top * vpxl_step + (*new_rect).left * hpxl_step);   // first pixel

    for(v=(*new_rect).top; v<=(*new_rect).bottom; v++)        // select row
    {
       for(h=(*new_rect).left; h<(*new_rect).right; h+=skip)    // select column
       {
          pxl_color = (unsigned long *) pxl_addr;              // get pixel location
          *pxl_color =  color;                          // change pixel color
          pxl_addr += (unsigned long)(hpxl_step * skip);   /  move to next pixel column
       }

       pxl_addr = base + (v * vpxl_step + (*new_rect).left * hpxl_step);       // next pixel row

    }
}
//--------------------------------------------------------------------------------
void my_draw_point(short x, short y, unsigned long color)
{
    unsigned long hpxl_step, vpxl_step;
    unsigned long *pxl_color;
    unsigned long pxl_addr, base;

    base = vd[0].base;

    hpxl_step = 0x00000004;     // horz memory increment
    vpxl_step = 0x00000d00;     // vert memory increment

    pxl_addr = base + y * vpxl_step + x * hpxl_step;
    pxl_color = (unsigned long *) pxl_addr;            // get pixel location
    *pxl_color =  color;                          // change pixel color

}
//--------------------------------------------------------------------------------
void my_draw_rect(Rect rect, unsigned long color)
{
    short x, y;

    y = rect.top;
    for(x = rect.left; x<rect.right; x++)
       my_draw_point(x, y, color);

    x = rect.right;
    for(y= rect.top; y < rect.bottom; y++)
       my_draw_point(x, y, color);

    y = rect.bottom;
    for(x = rect.left; x<= rect.right; x++)
       my_draw_point(x, y, color);

    x = rect.left;
    for(y= rect.top;  y < rect.bottom; y++)
       my_draw_point(x, y, color);

}
//--------------------------------------------------------------------------------
void my_draw_frame(Rect *fr, char fw)
{
       short x, y, left, top, right, bottom;

       // frame, left
       left = (*fr).left-fw; top = (*fr).top-fw; right = (*fr).left; bottom = (*fr).bottom+fw;
       for(x = left; x < right; x++)
       {
          for(y = top; y <= bottom; y++)
             my_draw_point(x, y, LEFT_FR);
       }

       // frame, right
       left = (*fr).right; top = (*fr).top-fw; right  = (*fr).right + fw; bottom = (*fr).bottom+fw;
       for(x = left; x<=right; x++)
       {
          for(y = top; y <= bottom; y++)
             my_draw_point(x, y, RIGHT_FR);
       }

       // frame, top
       left = (*fr).left-fw; top = (*fr).top-fw; right =(*fr).right + fw; bottom = (*fr).top;
       for(y = top; y<=bottom; y++)
       {
          for(x = left; x<=right; x++)
             my_draw_point(x, y, TOP_FR);
          left++;
          right--;
       }

       // frame, bottom
       left = (*fr).left; top = (*fr).bottom; right = (*fr).right; bottom = (*fr).bottom+fw;
       for(y = top; y<=bottom; y++)
       {
          for(x = left; x<=right; x++)
             my_draw_point(x, y, BOTTOM_FR);
          left--;
          right++;
       }
   }
//--------------------------------------------------------------------------------
void my_draw_frame_in(Rect *fr, char fw)
{
       short x, y, left, top, right, bottom;

       // frame, left
       left = (*fr).left; top = (*fr).top; right = (*fr).left + fw; bottom = (*fr).bottom;
       for(x = left; x < right; x++)
       {
          for(y = top; y <= bottom; y++)
             my_draw_point(x, y, LEFT_FR);
       }

       // frame, right
       left = (*fr).right-fw; top = (*fr).top; right = (*fr).right; bottom = (*fr).bottom;
       for(x = left; x<=right; x++)
       {
          for(y = top; y <= bottom; y++)
             my_draw_point(x, y, RIGHT_FR);
       }

       // frame, top
       left = (*fr).left; top = (*fr).top; right =(*fr).right; bottom = (*fr).top+fw;
       for(y = top; y<=bottom; y++)
       {
          for(x = left; x<=right; x++)
             my_draw_point(x, y, TOP_FR);
          left++;
          right--;
       }

       // frame, bottom
       left = (*fr).left+fw; top = (*fr).bottom-fw; right = (*fr).right-fw; bottom = (*fr).bottom;
       for(y = top; y<=bottom; y++)
       {
          for(x = left; x<=right; x++)
             my_draw_point(x, y, BOTTOM_FR);
```

```
        left--;
        right++;
    }
}
//-------------------------------------------------------------------------------
void my_draw_circle(circle *circle, unsigned long color)
{
  short x, y;
  short r = (*circle).radius;
  short left =  - (*circle).radius;
  short right =  (*circle).radius;

  for(x = left; x < right; x++)
  {
    y = (*circle).center.v + (short)pow(((double)(r * r) - (double)(x * x)), 0.5);
    my_draw_point(x+(*circle).center.h, y, color);
    y = (*circle).center.v - (short)pow(((double)(r * r) - (double)(x * x)), 0.5);
    my_draw_point(x+(*circle).center.h, y, color);
  }
}
//-------------------------------------------------------------------------------
void my_fill_circle(circle *circle, unsigned long color)
{
  short x, y, y_bottom, y_top;
  short r = (*circle).radius;
  short left =  - (*circle).radius;
  short right =  (*circle).radius;

  for(x = left;  x < right; x++)
  {
    y_bottom = (*circle).center.v + (short)pow(((double)(r * r) - (double)(x * x)), 0.5);
    y_top = (*circle).center.v - (short)pow(((double)(r * r) - (double)(x * x)), 0.5);
    for(y = y_top; y < y_bottom; y++)
      my_draw_point(x+(*circle).center.h, y, color);
  }

}
//-------------------------------------------------------------------------------
void my_draw_line(Point p1, Point p2, unsigned long color)
{
  short x, y, m, b;

  if(p2.h != p1.h)
  {
    m = (p2.v - p1.v) / (p2.h - p1.h);
    b = p2.v - m * p2.h;

    if(p1.h > p2.h)
    {
      for(x = p1.h; x < p2.h; x++)
      {
        y = m * x + b;
        my_draw_point(x, y, color);
      }
    }
    else if(p1.h < p2.h)
    {
      for(x = p2.h; x < p1.h; x++)
      {
        y = m * x + b;
        my_draw_point(x, y, color);
      }
    }
  }
  else
  {
    x = p1.h;
    if(p1.v < p2.v)
    {
      for(y = p1.v; y < p2.v; y++)
        my_draw_point(x, y, color);
    }
    else
    {
      for(y = p2.v; y < p1.v; y++)
        my_draw_point(x, y, color);
    }
  }
}
//-------------------------------------------------------------------------------
void my_draw_v_level(Rect level_rect, short level, short min, short max)
{
  float height = 0;
  int first_top = level_rect.top;

  HideCursor();
  if(level <= max && level >= min)
  {
    height = ((float)level / (max - min)) * (level_rect.bottom - level_rect.top);
    level_rect.top = level_rect.bottom - (int)height;
    my_fill_rect(&level_rect, BLACK, 1);

    level_rect.top = first_top;
    level_rect.bottom -= height;
    my_fill_rect(&level_rect, INSET_BG, 1);
  }
  ShowCursor();
}
//-------------------------------------------------------------------------------
void my_draw_h_bilevel(Rect level_rect, short level, short min, short max)
{
  int level_rect_mid, left_side, right_side;

  HideCursor();

  level_rect_mid = level_rect.left + 0.5 * (level_rect.right - level_rect.left);

  if(level >= -max && level <= max)
  {
    if(level < 0)
    {

      left_side = level_rect_mid + ((float) level / (max - min)) * 0.5 * (level_rect.right - level_rect.left);
      right_side = level_rect_mid;
    }
    else
    {
      left_side = level_rect_mid;
      right_side = level_rect_mid + ((float)level / (max - min)) * 0.5 * (level_rect.right - level_rect.left);
    }
  }
  else if (level > max)
  {
    left_side = level_rect_mid;
    right_side = level_rect.right;
  }
  else
  {
    left_side = level_rect.left;
    right_side = level_rect_mid;
  }
```

```
    my_fill_rect(&level_rect, INSET_BG, 1);
    level_rect.left = left_side;
    level_rect.right = right_side;
    my_fill_rect(&level_rect, BLACK, 1);

    ShowCursor();
}
// ---------------------------------------------------------------------------------------
void my_draw_v_bilevel(Rect level_rect, short level, short min, short max)
{
    short level_rect_mid, top_side, bottom_side;

    HideCursor();

    level_rect_mid = level_rect.top + 0.5 * (level_rect.bottom - level_rect.top);


    if(level >= -max && level <= max)
    {
        if(level < 0)
        {
            top_side = level_rect_mid + ((float) level / (max - min)) * 0.5 * (level_rect.bottom - level_rect.top);
            bottom_side = level_rect_mid;
        }
        else
        {
            top_side = level_rect_mid;
            bottom_side = level_rect_mid + ((float)level / (max - min)) * 0.5 * (level_rect.bottom - level_rect.top);
        }
    }
    else if (level < max)
    {
        bottom_side = level_rect_mid;
        top_side = level_rect.top;
    }
    else
    {
        bottom_side = level_rect.bottom;
        top_side = level_rect_mid;
    }

    my_fill_rect(&level_rect, INSET_BG, 1);
    level_rect.top = top_side;
    level_rect.bottom = bottom_side;
    my_fill_rect(&level_rect, BLACK, 1);

    ShowCursor();
}
// ---------------------------------------------------------------------------------------


void myprint(char info_type, char *message, int value)
{
    char print_string[100] = {0};

    if(value)
        sprintf(print_string, "%s %i", message, value);
    else
        sprintf(print_string, "%s", message);

    //switch(info_type)
    //{
    /  case status_line:
            EraseRect(&status_rect);
            MoveTo(status_rect.left, status_rect.top + 12);
            drawstring(print_string);
    /       break;
        //case info_line:
            //EraseRect(&info_rect);
            //MoveTo(info_rect.left, info_rect.top + 12);
            //drawstring(print_string);
            //break;
        //case error_line:
            //EraseRect(&error_rect);
            //MoveTo(error_rect.left, error_rect.top + 12);
            //drawstring(print_string);
            //SysBeep(60);
            //break;
    //}
}
// ------
float my_round(float number)
{
    if(ceil(number) < number + 0.5)
        number = ceil(number);
    else
        number = floor(number);

    return number;
}
// ------
// Obtain save file name
short get_save_file(unsigned char fileName[], StandardFileReply *outputReply)
{
    short           inFileRefNum;
    OSErr           fileError;
    short           oldVol;
    SFTypeList      textType = {'TEXT'};

    // Open the output file
        StandardPutFile ("\pSave settings in:", fileName, outputReply);
        if ((*outputReply).sfGood)
        {
            SetVol(NIL, (*outputReply).sfFile.vRefNum); // Make the destination volume current
            fileError = FSpCreate(&(*outputReply).sfFile, gfileCreator, gfileType, smSystemScript);
            switch(fileError)                          // Process result from File Manager
            {
                case noErr:
                    return 1;
                case dupFNErr:                         // File already exists, wipe it out
                    if ((fileError = FSpDelete(&(*outputReply).sfFile)) == noErr)
                    {
                        if ((fileError = FSpCreate(&(*outputReply).sfFile, gfileCreator,
                                gfileType, smSystemScript)) != noErr)
                        {
                            report_error(fileError);
                            FSClose (inFileRefNum);
                            SetVol(NIL, oldVol);
                            return 0;
                        } // end if != noErr
                        return 1;
                    } // end == noErr
                    else
                    {
                        report_error(fileError);
                        FSClose (inFileRefNum);
                        SetVol(NIL, oldVol);
                        return 0;
                    } // end else
                    break;  // end case dupFNErr
                default:
                    report_error(fileError);
            FSClose (inFileRefNum);          // Close the input file
            SetVol(NIL, oldVol);             // Restore original volume
            return 0;
          } // end switch
        } /* end if outputReply.sfGood */

      FSClose (inFileRefNum);
      SetVol(NIL, oldVol);
      return 0;                  /* Restore current volume */
}

// -------
// Open the input file

//short get_read_file()
//{
/  StandardGetFile(NIL, ONE_FILE_TYPE, textType, &inputReply);
/  if (inputReply.sfGood)
/  {
/     GetVol (NIL, &oldVol);          /* Save current volume */
/     if ((fileError = FSpOpenDF (&inputReply.sfFile, fsCurPerm, &inFileRefNum)) != noErr)
/     {
/        Report_Error(fileError);
/        return;
/     } /* end if error */
/  }
//}
//-----------------------------------------------------------------------------

void report_error(OSErr errorCode)
{
   unsigned char errNumString[8];

   NumToString((long) errorCode, errNumString);
   ParamText(errNumString, NIL, NIL, NIL);
   StopAlert(130, NIL);  // 130 = ERROR_BOX_ID
}

// ----------------------------------------------------------------------------
void write_line(short outFileRefNum, char *string, short tabs, short lines)
{
      unsigned char buffer;
      long amount = strlen(string);
      short i;

      FSWrite(outFileRefNum, &amount, string);

      buffer = TAB;
      amount = 1L;
      for(i = 0; i < tabs; i++)
         FSWrite(outFileRefNum, &amount, &buffer);

      buffer = NEW_LINE;
      for(i = 0; i < lines; i++)
         FSWrite(outFileRefNum, &amount, &buffer);
}
// ----------------------------------------------------------------------------
void write_control(short outFileRefNum, unsigned char control)
{
   long amount = 1L;
   FSWrite(outFileRefNum, &amount, &control);
}
// ----------------------------------------------------------------------------
void write_data(short outFileRefNum, char *string)
{
      long amount = strlen(string);
      FSWrite(outFileRefNum, &amount, string);
}
// ----------------------------------------------------------------------------
void tab(short outFileRefNum, short tab_count)
{
   short i;
   unsigned char control = TAB;
   long amount = 1L;

   for(i=0; i<tab_count; i++)
      FSWrite(outFileRefNum, &amount, &control);
}
// ----------------------------------------------------------------------------
void new_line(short outFileRefNum, short line_count)
{
   short i;
   unsigned char control = NEW_LINE;
   long amount = 1L;

   for(i=0; i<line_count; i++)
      FSWrite(outFileRefNum, &amount, &control);
}
// ----------------------------------------------------------------------------
```

```
//------------------------------------------//
/     taos_im_analy.h /
//------------------------------------------//

#include  <StdIO.h>
#include  <StdLib.h>
#include  <OSUtils.h>
#include <iostream.h>
#include  <QuickDraw.h>
#include  <Memory.h>
#include <FCntl.h>
#include <string.h>
#include <Dialogs.h>
#include "clipgrab.h"
```

```
#include "gauss.h"

unsigned long *lutp1, **luth1;
unsigned long *lutp2, **luth2;
unsigned long *lutp;

extern Rect image_map_finished_rect;
extern EventRecord    gEvent;
extern Rect image_map_hpixel_rect;
extern Rect image_map_vpixel_rect;
extern Rect image_map_hsetpixel_rect;
extern Rect image_map_vsetpixel_rect;
extern Rect image_map_hmeter_rect;
extern Rect image_map_vmeter_rect;
extern Rect next_map_set_rect;
extern Rect less_hmeter_rect;
extern Rect more_hmeter_rect;
extern Rect less_vmeter_rect;
extern Rect more_vmeter_rect;

void show_lut(unsigned long base);
int get_lut();
void do_image_map(short vd_number);
extern short mouse_in(Rect c_rect, Point click);
extern void do_text_rect(Rect rect,  char *string, char align);
extern void gaussian_solve_3(double in[], double out[]);

//----------------------------------------------------------------------------------
// define region of interest (roi)

  class roi
  {
    private:
      unsigned long pxl_addr, hpxl_step, vpxl_step;
      short minh, maxh, hstep, minv, maxv, vstep;
      float minhm, maxhm, minvm, maxvm;
      short noise;
    public:
      void set_roi(char *string, float set_width, float *band_adj, unsigned long base,
          float voffsetm);
      int analyze_roi(int offset, int show_pixel);
      void adjust_roi(short v_shift, unsigned long base);
      short find_max_vpixel(void);
      short find_min_vpixel(void);
      short find_max_hpixel(void);
      void add_roi(roi *zone);
      float roi_nearside(void);
      float roi_farside(void);
      short roi_horz_skip(void);
      short roi_vert_skip(void);
      short roi_noise(void );
      void edit_roi_nearside(float value, unsigned long base, float *band_adj);
      void edit_roi_farside(float value);
      void edit_roi_h_skip(int value);
      void edit_roi_v_skip(int value);
      void edit_roi_noise(int value);
      void reset_roi(short v_shift, unsigned long base);
      void edit_control_width(unsigned long control_increment);
      void shift_band(float roi_voffsetm);
      void make_big_zone(Rect *rect);
  };

//----------------------------------------------------------------------------------
// convert roi's from meters to pixels
void roi::set_roi(char *string, float set_width, float *band_adj, unsigned long base,
  float voffsetm)
{
  char *p;
  //float *band_adj;
  short band, dx, dy, nse;
  short vd_number = 0;
  short voffsetp = 0;

  p = strtok(string, " \t\n");
    band = atoi(p);
  p = strtok(NULL, " \t\n");
    minhm = atof(p);
  p = strtok(NULL, " \t\n");
    maxhm = atof(p);
  p = strtok(NULL, " \t\n");
    dx = atoi(p);
  p = strtok(NULL, " \t\n");
    dy = atoi(p);
  p = strtok(NULL, " \t\n");
    nse = atoi(p);

  minvm = 0;
  maxvm = set_width;

  // top of roi
  voffsetp = ((float)(vd[0].disp_rect.bottom - vd[0].disp_rect.top)/2)-my_round(voffsetm *
    (vd[vd_number].v_coef0 + (vd[vd_number].v_coef1 * minhm) + (vd[vd_number].v_coef2 *
    minhm * minhm)));

  // left side of roi
  minh = my_round(vd[vd_number].h_coef1 * minhm + (vd[vd_number].h_coef2 * minhm * minhm));

  // right side of roi
  maxh = my_round(vd[vd_number].h_coef1*maxhm + (vd[vd_number].h_coef2 * maxhm * maxhm));

  hstep = dx;                                          // horz pixels skipped
  minv = my_round(minvm * (vd[vd_number].v_coef0 + (vd[vd_number].v_coef1 * minhm) +
    (vd[vd_number].v_coef2 * minhm * minhm)));         // top of roi
  minv += voffsetp;
  maxv = my_round(2+maxvm * (vd[vd_number].v_coef0 + (vd[vd_number].v_coef1 * minhm) +
    (vd[vd_number].v_coef2 * minhm * minhm)));         // bottom of roi
  maxv += voffsetp;
  vstep = dy;                                          // vert pixels skipped
  noise = nse;                                        // pixels allowed for noise
  hpxl_step = dx * 0x00000004;                           // horz memory increment
  vpxl_step = 0x00000d00;                                // vert memory increment
  pxl_addr = base + ( minh * hpxl_step/dx);              // first pixel (horz)
  *band_adj = (vd[vd_number].v_coef0 + (vd[vd_number].v_coef1 * minhm) +
    (vd[vd_number].v_coef2 * minhm * minhm)) / RAVENS;    // pixel/raven
}
//----------------------------------------------------------------------------------

// analyze region in video image (specfied by borders, pixel density, noise allowance)

int roi::analyze_roi(int offset, int show_pixel)
{
  register int v, h, pixel_count=0;
  unsigned long pxl_color, *pxl_value;
  unsigned long new_pxl_addr, start_pxl;

  start_pxl = pxl_addr + offset * vpxl_step;

  for(v=minv; v<maxv; v+=vstep)                        // vertical scan
  {
    new_pxl_addr = v * vpxl_step + start_pxl;          // left starting pixel
    for(h=minh; h<maxh; h+=hstep)                      // horizontal scan
    {
         pxl_value = (unsigned long *) new_pxl_addr;  // get pixel value address
         pxl_color = *pxl_value & 0x00ffffff;        // extract 32 bit color
         if(((1L<<(pxl_color & 0x0000001F)) & *(lutp + (pxl_color >> 5))) &&
            ((pxl_color & 0x0000ff00) > ((pxl_color & 0x000000ff) << 8)))
         {
            pixel_count++;                   // record number of target color pixels
            if(show_pixel)
               *pxl_value = RED;
         }
         else if(show_pixel)
            *pxl_value = BLACK;

         new_pxl_addr += (unsigned long)hpxl_step;    //  move to next pixel column
      }
   }
   return (pixel_count>noise) ? 1: 0;                // 1 if target, 0 if not
}

// end of analyze region

//------------------------------------------------------------------------------------
short roi::find_min_vpixel(void)
{
   return minv;
}
//------------------------------------------------------------------------------------
short roi::find_max_vpixel(void)
{
   return maxv;
}
//------------------------------------------------------------------------------------
short roi:: find_max_hpixel(void)
{
   return maxh;
}
//------------------------------------------------------------------------------------
float roi::roi_nearside(void)
{
   return minhm;
}
//------------------------------------------------------------------------------------
float roi::roi_farside(void)
{
   return maxhm;
}
//------------------------------------------------------------------------------------
short roi::roi_horz_skip(void)
{
   return hstep;
}
//------------------------------------------------------------------------------------
short roi::roi_vert_skip(void)
{
   return vstep;
}
//------------------------------------------------------------------------------------
short roi::roi_noise(void)
{
   return noise;
}
//------------------------------------------------------------------------------------
void roi::add_roi(roi *zone)
{
   minh = (*zone).minh;                          // left side of roi
   maxh = (*zone).maxh;                             // right side of roi
   hstep = (*zone).hstep;                      // horz pixels skipped
   minv = (*zone).maxv;
   maxv = (*zone).maxv + ((*zone).maxv - (*zone).minv);
   vstep = (*zone).vstep;                          // vert pixels skipped
   noise = (*zone).noise;                          // pixels allowed for noise
   hpxl_step = (*zone).hpxl_step;                     // horz memory increment
   vpxl_step = (*zone).vpxl_step;                     // vert memory increment
   pxl_addr = (*zone).pxl_addr;                       // first pixel
   minhm = (*zone).minhm;                                // left side of roi
   maxhm = (*zone).maxhm;
   minvm = (*zone).maxvm;
   maxvm = (*zone).maxvm + ((*zone).maxvm - (*zone).minvm);
}
//------------------------------------------------------------------------------------
void roi::adjust_roi(short v_shift, unsigned long base)
{
   Rect  roi_rect;

   minv += VD_1_WINDOW_TOP + 4 + v_shift;
   maxv += VD_1_WINDOW_TOP + 4 + v_shift;
   minh += VD_1_WINDOW_LEFT + 4;
   maxh += VD_1_WINDOW_LEFT + 4;
   pxl_addr = base + ( minh * hpxl_step/hstep);

   SetRect(&roi_rect, minh, minv, maxh, maxv);
   my_fill_rect(&roi_rect, 0x00ff0000, 2);
}
//------------------------------------------------------------------------------------
void roi::reset_roi(short v_shift, unsigned long base)
{
   minv -=  v_shift;
   maxv -=  v_shift;
   minh -= VD_1_WINDOW_LEFT + 4;
   maxh -= VD_1_WINDOW_LEFT + 4;
   pxl_addr = base + ( minh * hpxl_step/hstep);
}
//------------------------------------------------------------------------------------
void roi::shift_band(float voffsetm)
{
   short voffsetp;
   short vd_number = 0;

   voffsetp = ((float)(vd[0].disp_rect.bottom - vd[0].disp_rect.top)/2)-my_round(voffsetm *
      (vd[vd_number].v_coef0 + (vd[vd_number].v_coef1 * minhm) + (vd[vd_number].v_coef2 *
      minhm * minhm)));

   minv += voffsetp;
   maxv += voffsetp;

}

//------------------------------------------------------------------------------------
void roi::edit_roi_nearside(float value, unsigned long base, float *band_adj)
{
   minhm = value;
   short vd_number = 0;

   minh = my_round(vd[vd_number].h_coef1 * minhm + (vd[vd_number].h_coef2 * minhm * minhm)); // left side
of roi

   minv = my_round(minvm * (vd[vd_number].v_coef0 + (vd[vd_number].v_coef1 * minhm) +
      (vd[vd_number].v_coef2 * minhm * minhm)));            // top of roi
   maxv = my_round(2+maxvm * (vd[vd_number].v_coef0 + (vd[vd_number].v_coef1 * minhm) +
      (vd[vd_number].v_coef2 * minhm * minhm)));            // bottom of roi
   pxl_addr = base + ( minh * hpxl_step/hstep);            // first pixel (horz)
   *band_adj = (vd[vd_number].v_coef0 - (vd[vd_number].v_coef1 * minhm) +
      (vd[vd_number].v_coef2 * minhm * minhm)) / RAVENS;        // pixel/raven
```

```
}
//--------------------------------------------------------------------------------
void roi::edit_roi_farside(float value)
{
   short vd_number = 0;

   maxhm = value;
   maxh = my_round(vd[vd_number].h_coef1 * maxhm + (vd[vd_number].h_coef2 * maxhm * maxhm));  // right
side of roi

}
//--------------------------------------------------------------------------------
void roi::edit_roi_h_skip(int value)
{
   hstep = value;
   hpxl_step = value * 0x00000004;
}
//--------------------------------------------------------------------------------
void roi::edit_roi_v_skip(int value)
{
   vstep = value;
}
//--------------------------------------------------------------------------------
void roi::edit_roi_noise(int value)
{
   noise = value;
}
//--------------------------------------------------------------------------------
void roi::edit_control_width(unsigned long control_increment)
{
   maxvm =  (float)control_increment / RAVENS;
   maxv = my_round(2+maxvm * (450.7 - (115.0 * minhm) + (8.2 * minhm * minhm)));  // bottom of roi
}
//--------------------------------------------------------------------------------
void roi::make_big_zone(Rect *rect)
{
   short vd_number = 0;

   minh = (*rect).left;
   minv = (*rect).top;
   maxh = (*rect).right;
   maxv = (*rect).bottom;

   hstep = 1;
   vstep = 1;
   noise = 1;
   hpxl_step = hstep * 0x00000004;
   vpxl_step = vstep * 0x00000d00;
   pxl_addr = vd[vd_number].base+(minh * hpxl_step/hstep);
}
//--------------------------------------------------------------------------------
// show lut colors

void show_lut(unsigned long base)
{
   short          v, h;
   unsigned long  *pxl_value, lut_color=0x00000000, hpxl_step;
   short          minv, maxv, vstep, minh, maxh, hstep, hflag = 0;
   unsigned long  vpxl_step, lpxl_addr; //, hgrid_step;
   EventRecord    *event;
   char           key;
   short          delay_loop;
   WindowPtr         LUT_Window, current_window;
   Rect              LUT_rect;

   current_window = FrontWindow();
   HideWindow(current_window);

   SetRect(&LUT_rect, 50, 50, 800, 620);
   LUT_Window = NewCWindow(NULL, &LUT_rect, "\pColor Look Up Table [CLUT]", true, noGrowDocProc,
(WindowPtr)(-1), false, NULL);

   minv = 20;
   maxv = minv+256;
   vstep= 1;

   minh = 20;
   maxh = minh+256;
   hstep = 1;

   hpxl_step = hstep * 0x00000004;         // horz memory increment
   vpxl_step = 0x00000d00;                 // vert memory increment

   lpxl_addr =  base + (minv * vpxl_step + minh * hpxl_step);   // first pixel

   while(lut_color <= 0x00ffffff)
   {
   for(v=minv; v<maxv; v+=vstep)            // select row
   {
     for(h=minh; h<maxh; h+=hstep)          // select column
     {
        pxl_value = (unsigned long *) lpxl_addr;  // get pixel value address
        if(!((1L<<(lut_color & 0x0000001F)) & *(lutp + (lut_color >> 5))))
           *pxl_value = 0x00000000;
        else
           *pxl_value = lut_color;
        if((h-minh<1)&&(v-minv<=1)||(((h-minh<1)||(v-minv<=1))&&(lut_color>=0x00ff0000)))
             *pxl_value = 0x00ffffff;           // check for boundary
        lpxl_addr += (unsigned long)hpxl_step;   //  move to next pixel column
        lut_color++;
     }

     lpxl_addr = base + (v * vpxl_step + minh * hpxl_step);       // next pixel row
   }

   minh++;     // provide "3-D" offset
   minv++;
   maxh++;
   maxv++;

   if(lut_color >= 0x00ffffff)  // pause if end of color space reached
     delay_loop = 1;

   do
   {
     GetNextEvent(everyEvent, event);
     switch ( event->what )
     {
        case keyDown:                          // keyboard input
        key = event->message & charCodeMask;
          switch (key)
          {
            case 'p':
            {
              delay_loop = (delay_loop) ? 0: 1;
              if(delay_loop)
              {
                lpxl_addr = base + (minv * vpxl_step + minh * hpxl_step);    // first pixel
                v=minv;                        // select row
                for(h=minh; h<maxh; h+=hstep)       // select column
                {
                  pxl_value = (unsigned long *) lpxl_addr;  // get pixel value address
```

```
                    *pxl_value = 0x00ffffff;
                    lpxl_addr += (unsigned long)hpxl_step;    // move to next pixel column
                  }

                  lpxl_addr = base + (minv * vpxl_step + minh * hpxl_step);    // first pixel
                  h=minh;                                   // select row
                  for(v=minh; v<maxh; v+=vstep)                   // select column
                  {
                    pxl_value = (unsigned long *) lpxl_addr;  // get pixel value address
                    *pxl_value = 0x00ffffff;
                    lpxl_addr = base + (v * vpxl_step + minh * hpxl_step);     // next pixel row
                  }
                }
                break;
              }
              case 'c':
              {
                SysBeep(1);
                //grab_to_clip(LUT_rect.left, LUT_rect.top, LUT_rect.right-LUT_rect.left, LUT_rect.bottom-
LUT_rect.top);
                grab_to_clip(LUT_rect.left, LUT_rect.top, LUT_rect.left+400, LUT_rect.top+400);

              }
              break;
            }
          }
      }while(delay_loop);

      lpxl_addr = base + (minv * vpxl_step + minh * hpxl_step); // start of next rectangle

  }

  DisposeWindow(LUT_Window);

  ShowWindow(current_window);

}

// end of lut display
//-----------------------------------------------------------------------------------------
// getlook-up table of plant colors

int get_lut()
{
  FILE *fid;
  lutp1 = nil; luth1 = nil;
  lutp2 = nil; luth2 = nil;

  // get standard CLUT data
  fid = fopen("CLUT.STANDARD", "rb"); // open CLUT
  if(fid)
  {
    luth1 = (unsigned long **) NewHandle(524288 * (sizeof(long)));
    if(luth1 != nil)
    {

      MoveHHi((Handle) luth1);
      HLock( (Handle) luth1 );
      lutp1 = (unsigned long *) *luth1;
      fread(lutp1,sizeof(long),524288,fid);
    }
    else
    {
      SysBeep(1);
      myprint(status_line, "Standard Look-up table file error", 0);
      pause(5.0);
      exit(0);
    }
    fclose(fid);          // all o.k.
  }
  else
    return(1);            // error - return 1

  // get shadow CLUT data
  fid = fopen("CLUT.SHADOWS", "rb"); // open CLUT
  if(fid)
  {
    luth2 = (unsigned long **) NewHandle(524288 * (sizeof(long)));
    if(luth2 != nil)
    {

      MoveHHi((Handle) luth2);
      HLock( (Handle) luth2 );
      lutp2 = (unsigned long *) *luth2;
      fread(lutp2,sizeof(long),524288,fid);
    }
    else
    {
      SysBeep(1);
      myprint(status_line, "Look-up table file error", 0);
      pause(5.0);
      exit(0);
    }
    fclose(fid);
    return(0);            // all o.k. - return 0
  }
  else
    return(1);            // error - return 1

} // end of get_lut

//-----------------------------------------------------------------------------------------
//void do_image_map(unsigned long base, Rect vd_rect, short refnum)
void do_image_map(short vd_number)
{
  unsigned long hpxl_step, vpxl_step;
  short x, y;
  short h[10] = {0}, v[10] = {0};
  float hmeter[10] = {0.0}, vmeter[10] = {0.0};
  unsigned long color = 0x00dd0000;
  Point mouse_down_loc;
  char string[50] = {0};
  circle   mark[10][3], cursor_pos;
  short i, j, set;
  short first_vd_right = 0;
  short width, height;

  float a[3][3] = {0.0}, b[3] = {0.0}, c[3] = {0.0};

  set = 0;

  for(i = 0; i<10; i++ )
    for(j=0; j<2; j++)
    {
      mark[i][j].radius = 3;
      mark[i][j].center.h = 0;
      mark[i][j].center.v = 0;
    }

  cursor_pos.radius = 3;

  hpxl_step = 0x00000004;     // horz memory increment
```

```
vpxl_step = 0x00000d00;     // vert memory increment

first_vd_right = vd[vd_number].image_rect.right;
vd[vd_number].image_rect.right = vd[vd_number].disp_rect.right;
width = vd[vd_number].image_rect.right - vd[vd_number].image_rect.left;
height = vd[vd_number].image_rect.bottom - vd[vd_number].image_rect.top;
vd_set_rect(&vd[vd_number].params, &vd[vd_number].image_rect, width, height, 1);

cont_frame(&vd[vd_number].params, DONTWAIT);

i = 0;

sprintf(string, "set number: %i", set);
myprint(status_line, string, 0);

while(!mouse_in(image_map_finished_rect, mouse_down_loc)  && set < 10)
{

  WaitNextEvent(everyEvent, &gEvent, LONG_NAP, NO_CURSOR);
  if(gEvent.what == mouseDown)
    GetMouse(&mouse_down_loc);

  if(mouse_in(vd[vd_number].disp_rect, gEvent.where))
  {
    x = gEvent.where.h;
    y = gEvent.where.v;

    HideCursor();

    grab_frame(1, vd[vd_number].refnum, WAIT);      // first slot with vd

    if(i == 0)
      mark[set][0].center.h = x;

    mark[set][i].center.v = y;

    cursor_pos.center.h = x;
    cursor_pos.center.v = y;

    my_fill_circle(&cursor_pos, GREEN);

    for(j=0; j<set; j++)
    {
      my_fill_circle(&mark[j][0], BLUE);
      my_fill_circle(&mark[j][1], BLUE);
      my_draw_line(mark[j][0].center, mark[j][1].center, BLUE);
    }

    my_fill_circle(&mark[set][0], RED);

    if(i>=1)
    {
      my_fill_circle(&mark[set][1], RED);
      my_draw_line(mark[set][0].center, mark[set][1].center, RED);
    }

    sprintf(string, "%i", i);
    myprint(status_line, string, 0);

    sprintf(string, "h pixel: %hi", x - vd[vd_number].disp_rect.left);
    my_fill_rect(&image_map_hpixel_rect, WHITE, 1);
    do_text_rect(image_map_hpixel_rect, string, 0);
    sprintf(string, "v pixel: %hi", y - vd[vd_number].disp_rect.top);
    my_fill_rect(&image_map_vpixel_rect, WHITE, 1);
    do_text_rect(image_map_vpixel_rect, string, 0);

    if(gEvent.what == mouseDown)
    {
      if(i == 1)
      {
        h[set] = mark[set][0].center.h - vd[vd_number].disp_rect.left;
        v[set] = mark[set][1].center.v - mark[set][0].center.v;

        if((vd[vd_number].h_coef1 * vd[vd_number].h_coef1 + 4 * vd[vd_number].h_coef2 * (float)h[set]) >
0)
          hmeter[set] = (-vd[vd_number].h_coef1 + sqrt(vd[vd_number].h_coef1 *
            vd[vd_number].h_coef1 + 4 * vd[vd_number].h_coef2 * (float)h[set]))/ (2 *
vd[vd_number].h_coef2);
        else
          hmeter[set] = 3.0;

        vmeter[set] = (float)(v[set]) / (vd[vd_number].v_coef0 + vd[vd_number].v_coef1 *
          hmeter[set] + vd[vd_number].v_coef2 * hmeter[set] * hmeter[set]);

        sprintf(string, "h pixel: %hi", h[set]);
        my_fill_rect(&image_map_hsetpixel_rect, WHITE, 1);
        do_text_rect(image_map_hsetpixel_rect, string, 0);
        sprintf(string, "v pixel: %hi", v[set]);
        my_fill_rect(&image_map_vsetpixel_rect, WHITE, 1);
        do_text_rect(image_map_vsetpixel_r ect, string, 0);

        sprintf(string, "h meter: %5.2f", hmeter[set]);
        my_fill_rect(&image_map_hmeter_rect, WHITE, 1);
        do_text_rect(image_map_hmeter_rect, string, 0);
        sprintf(string, "v meter: %5.2f", vmeter[set]);
        my_fill_rect(&image_map_vmeter_rect, WHITE, 1);
        do_text_rect(image_map_vmeter_rect, string, 0);
      }

      if(i < 2)
        i++;

      mark[set][1].center.h = mark[set][0].center.h;

    }
  }
  else      // not in image rectangle
  {
    ShowCursor();
    i = 0;
    if(gEvent.what == mouseDown)
    {
      if(mouse_in(next_map_set_rect, gEvent.where))
      {
        set++;
        sprintf(string, "set number: %i", set);
        myprint(status_line, string, 0);
      }
      do
      {
        if(mouse_in(more_hmeter_rect, gEvent.where))
        {
          hmeter[set] += 0.01;
          sprintf(string, "h meter: %5.2f", hmeter[set]);
          my_fill_rect(&image_map_hmeter_rect, WHITE, 1);
          do_text_rect(image_map_hmeter_rect, string, 0);
        }
        if(mouse_in(less_hmeter_rect, gEvent.where))
        {
          hmeter[set] -= 0.01;
          sprintf(string, "h meter: %5.2f", hmeter[set]);
          my_fill_rect(&image_map_hmeter_rect, WHITE, 1);
```

```
            do_text_rect(image_map_hmeter_rect, string, 0);
         }

         if(mouse_in(more_vmeter_rect, gEvent.where))
         {
            vmeter[set] += 0.01;
            sprintf(string, "v meter: %5.2f", vmeter[set]);
            my_fill_rect(&image_map_vmeter_rect, WHITE, 1);
            do_text_rect(image_map_vmeter_rect, string, 0);
         }
         if(mouse_in(less_vmeter_rect, gEvent.where))
         {
            vmeter[set] -= 0.01;
            sprintf(string, "v meter: %5.2f", vmeter[set]);
            my_fill_rect(&image_map_vmeter_rect, WHITE, 1);
            do_text_rect(image_map_vmeter_rect, string, 0);
         }
         pause(0.1);
      }while(StillDown());
   }
  }
 }
grab_frame(1, vd[vd_number].refnum, DONTWAIT);

// do curve fit
  if(set >= 3)
  {
    for(i = 0; i < set; i++)
    {
      a[0][0] ++;
      a[0][1] += hmeter[i];
      a[0][2] += hmeter[i] * hmeter[i];
      b[0] += h[i];
      a[1][0] += hmeter[i];
      a[1][1] += hmeter[i] * hmeter[i];
      a[1][2] += hmeter[i] * hmeter[i] * hmeter[i];
      b[1] += hmeter[i] * h[i];
      a[2][0] += hmeter[i] * hmeter[i];
      a[2][1] += hmeter[i] * hmeter[i] * hmeter[i];
      a[2][2] += hmeter[i] * hmeter[i] * hmeter[i] * hmeter[i];
      b[2] += hmeter[i] * h[i] * hmeter[i];
    }

    gauss_solve(a, b, c);

    vd[vd_number].h_coef1 = c[1];
    vd[vd_number].h_coef2 = c[2];

    sprintf(string, "h: %4.2f, %4.2f, %4.2f", c[0], c[1], c[2]);
    myprint(status_line, string, 0);

    pause(5.0);

    for(i = 0; i < 3; i++)
    {
      a[0][i] = 0;
      a[1][i] = 0;
      a[2][i] = 0;
      b[i] = 0;
      c[i] = 0;
    }

    for(i = 0; i < set; i++)
    {
      a[0][0] ++;
      a[0][1] += hmeter[i];
      a[0][2] += hmeter[i] * hmeter[i];
      b[0] += v[i]/vmeter[i];
      a[1][0] += hmeter[i];
      a[1][1] += hmeter[i] * hmeter[i];
      a[1][2] += hmeter[i] * hmeter[i] * hmeter[i];
      b[1] += hmeter[i] * v[i]/vmeter[i];
      a[2][0] += hmeter[i] * hmeter[i];
      a[2][1] += hmeter[i] * hmeter[i] * hmeter[i];
      a[2][2] += hmeter[i] * hmeter[i] * hmeter[i] * hmeter[i];
      b[2] += hmeter[i] * v[i]/vmeter[i] * hmeter[i];
    }

    gauss_solve(a, b, c);

    vd[vd_number].v_coef0 = c[0];
    vd[vd_number].v_coef1 = c[1];
    vd[vd_number].v_coef2 = c[2];

    sprintf(string, "v: %4.2f, %4.2f, %4.2f", c[0], c[1], c[2]);
    myprint(status_line, string, 0);

    pause(5.0);
  }
  //vd[vd_number].image_rect.right = zone[0][band_count-1].find_max_hpixel();
  vd[vd_number].image_rect.right = first_vd_right;
  vd_set_rect(&vd[vd_number].params, &vd[vd_number].image_rect, vd[vd_number].width,
vd[vd_number].height, 1);
}

//-----------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------

//------------------------------------------//
/    mac_ini.h   /
//------------------------------------------//
```

```
MenuHandle      gmyMenus[LAST_MENU + 1];   // handle to menus

Boolean initialize_mac(void);
unsigned char GoGetRect(short rectID, Rect *theRect);

   Rect data_rect;


// initialize macintosh

Boolean initialize_mac(void)
{
   short i;

   MaxApplZone();
   MoreMasters();
   MoreMasters();
   MoreMasters();
   MoreMasters();
   MoreMasters();
   MoreMasters();
   MoreMasters();
   MoreMasters();
   MoreMasters();
   MoreMasters();
   InitGraf(&qd.thePort);
   InitFonts();
   FlushEvents(everyEvent, 0);
   InitWindows();
   InitMenus();
   TEInit();
   InitDialogs(NIL);

   for(i = APPLE_MENU; i<(APPLE_MENU + LAST_MENU); i++)
   {
      gmyMenus[(i-RESOURCE_ID)] = GetMenu(i);
      if(gmyMenus[(i-RESOURCE_ID)]==NIL)
         return FALSE;
   }

   AppendResMenu(gmyMenus[(APPLE_MENU - RESOURCE_ID)], 'DRVR');

   for (i=APPLE_MENU; i<(APPLE_MENU + LAST_MENU); i++)
      InsertMenu(gmyMenus[(i - RESOURCE_ID)], APPEND_MENU);

   DrawMenuBar();
   InitCursor();

   return TRUE;

}  // end of intialize


// GoGetRect
unsigned char GoGetRect(short rectID, Rect *theRect)
//short rectID;
//Rect *theRect;
{
 Handle resource;

  resource = GetResource('RECT', rectID);
  if (resource != nil)
  {
   *theRect = **((Rect**) resource);
   return(true);
  }
 else
   return(false);
} /* End of GoGetRect */



//-----------------------------------------//
/    taos_valves.h    //
//-----------------------------------------//

//--------------------------------------------------------------------------------
// taos_valves.h

/ Written by Chris Tauzer
/ Created: January 1995
/ Modified:  November 1995

/  This is the valve control header file for the Target Activated Offset
// Sprayer (TAOS) developed by UCD BAE department in conjunction with
// Caltrans (AHMCT).  It is designed to work in conjunction with ODC5
// digital output relays mounted on PB-8H backplanes, and controlled from
// a National Instruments Lab-NB board.

//--------------------------------------------------------------------------------

#include "lab_nb_control.h"  // National Instrument Lab-NB control functions
#include <string.h>

#define VALVES_OFF  0x00000000

extern void show_signal(char band, char front_plant, char back_plant, unsigned long color);
extern short band_count;
extern void do_overspeed_warning(char warning_type);

extern short set_count;

//--------------------------------------------------------------------------------
// ----- valve class -----

class valves
{
  private:
    float dist, tof_adj, f_lead, f_lag, d_lead, d_lag, expand_norm, expand_tang;

    short number, band_in, band_out, valve_status;
  public:
    void set_valve(char *string);
    Boolean calculate_valve_signal(unsigned long plants[], float speed, char base_bit,
      int valve_action_increment);
    float get_dist();
    float get_tof();
    float get_f_lead();
    float get_f_lag();
    float get_d_lead();
    float get_d_lag();
    short get_band_in();
    short get_band_out();
    short get_status();
    float get_expand_norm();
    float get_expand_tang();
    float get_extend();
    void edit_dist(float new_dist);
    void edit_tof(float new_tof);
    void edit_f_lead(float new_f_lead);
    void edit_f_lag(float new_f_lag);
```

```
      void edit_d_lead(float new_d_lead);
      void edit_d_lag(float new_d_lag);
      void edit_band_in(short new_band_in);
      void edit_band_out(short new_band_out);
      void edit_expand_norm(float new_coef);
      void edit_expand_tang(float new_coef);

      void set_status(short valve_status);
};
//------------------------------------------------------------------------------------
// ----- set up valve configuration ----

void valves::set_valve(char *string)
[
   char *p;
   char new_string[100] = [0];

   // retrieve coefficients from string passed in from file
      p = strtok(string, " \t\n");        // find first token
      number = atoi(p);                    // valve number
      p = strtok( NULL, " \t\n");
      dist = atof(p) * RAVENS;             // distance from video to spray
      p = strtok( NULL, " \t\n");
      tof_adj = atof(p) * TICKS_PER_SEC;  // time of flight adjustment
      p = strtok( NULL, " \t\n");
      f_lead =atof(p) * RAVENS;            // fixed lead
      p = strtok( NULL, " \t\n");
      f_lag = atof(p) * RAVENS;            // fixed l ag
      p = strtok( NULL, " \t\n");
      d_lead =atof(p) * TICKS_PER_SEC;     // dynamic lead (speed sensitive)
      p = strtok( NULL, " \t\n");
      d_lag = atof(p) * TICKS_PER_SEC;    // dynamic lag (speed sensitive)
      p = strtok( NULL, " \t\n");
      band_in = atoi(p);                  // adjustment into near bands
      p = strtok( NULL, " \t\n");
      band_out = atoi(p);                 // adjustment into far bands
      p = strtok( NULL, " \t\n");
      expand_norm = atof(p);               // pattern expansion adj. (normal)
      p = strtok( NULL, " \t\n");
      expand_tang  = atof(p);              // pattern expansion adj. (tangent)

   // start with valves in auto mode
      valve_status = AUTO;
]

//------------------------------------------------------------------------------------
// calculate valve control signal based on plants detected

Boolean valves::calculate_valve_signal(unsigned long plants[], float speed, char base_bit,
   int valve_action_increment)
{
   // declare variables
      short band, back_plant, front_plant, first_band, last_band;
      unsigned long valve_signal=0;
      static char last_front_plant[MAX_BAND+1], last_back_plant[MAX_BAND+1];
      char string[100] = {0};

   // return if in manual mode
      if(valve_status == MANUAL_ON)
         return 1;
      if(valve_status == MANUAL_OFF)
         return 0;

   // calculate start and end of bits that we are interested in
      back_plant = ( - base_bit) + (((dist  + f_lag - ((tof_adj - d_lag) * speed))
         /valve_action_increment));
      front_plant = ( - base_bit) + (((dist - f_lead - ((tof_adj + d_lead) * speed))
         /valve_action_increment));

   // get bands to work with for this valve
      first_band = band_in;
      last_band = band_out;

   // expand pattern if expand option on
      if(status.expand)
      {
         back_plant += expand_tang;
         front_plant -= expand_tang;
         first_band -= expand_norm;
         last_band += expand_norm;
      }

      // watch for over-run of data
      if(front_plant < set_count || front_plant > 31 || back_plant < set_count || back_plant > 31)
      {
         do_overspeed_warning(2);
         return(1);                    // spray anyway just in case
      }

   // adjust first and last bands to fit within out data
      while(first_band < 0)            // we don't have bands less than 0
            first_band++;

      while(last_band >= band_count)       // can't look further out than the last one
            last_band--;

   // determine if the valve will need to spray or not
      for(band = first_band; band <= last_band; band++)
      {
         // the following zeroes extraneous information in plant data
         valve_signal += (( plants[band]  << (31-back_plant)) >> (31-back_plant))
            >> front_plant;

         // display the data if desired
         if(status.show_data)
         {
            show_signal(band, last_front_plant[band]-1, last_back_plant[band]-1
               , 0x00000000);
            show_signal(band, front_plant-1, back_plant-1, 0x00ff0000);
            last_front_plant[band] = front_plant;
            last_back_plant[band] = back_plant;
         }
      }

   // if plant information is non-zero, return a 1
      if(valve_signal)
         return(1);
      else
         return(0);
}

//------------------------------------------------------------------------------------
float valves::get_dist()
{
   return dist / RAVENS;
}
//------------------------------------------------------------------------------------
float valves::get_tof()
{
   return tof_adj / TICKS_PER_SEC;
}
//------------------------------------------------------------------------------------
```

```
float valves::get_f_lead()
{
   return f_lead / RAVENS;
}
//-------------------------------------------------------------------------------
float valves::get_f_lag()
{
   return f_lag / RAVENS;
}
//-------------------------------------------------------------------------------
float valves::get_d_lead()
{
   return d_lead / TICKS_PER_SEC;
}
//-------------------------------------------------------------------------------
float valves::get_d_lag()
{
   return d_lag / TICKS_PER_SEC;
}
//-------------------------------------------------------------------------------
short valves::get_band_in()
{
   return band_in;
}
//-------------------------------------------------------------------------------
short valves::get_band_out()
{
   return band_out;
}
//-------------------------------------------------------------------------------
float valves::get_expand_norm()
{
   return expand_norm;
}
//-------------------------------------------------------------------------------
float valves::get_expand_tang()
{
   return expand_tang;
}
//-------------------------------------------------------------------------------
short valves::get_status()
{
   return valve_status;
}
//-------------------------------------------------------------------------------
void valves::edit_dist(float new_dist)
{
   dist = new_dist * RAVENS;
}
//-------------------------------------------------------------------------------
void valves::edit_tof(float new_tof)
{
   tof_adj = new_tof * TICKS_PER_SEC;
}
//-------------------------------------------------------------------------------
void valves::edit_f_lead(float new_f_lead)
{
   f_lead = new_f_lead * RAVENS;
}
//-------------------------------------------------------------------------------
void valves::edit_f_lag(float new_f_lag)
{
   f_lag = new_f_lag * RAVENS;
}
//-------------------------------------------------------------------------------
void valves::edit_d_lead(float new_d_lead)
{
   d_lead = new_d_lead * TICKS_PER_SEC;
}
//-------------------------------------------------------------------------------
void valves::edit_d_lag(float new_d_lag)
{
   d_lag = new_d_lag * TICKS_PER_SEC;
}
//-------------------------------------------------------------------------------
void valves::edit_band_in(short new_band_in)
{
   band_in = new_band_in;
}
//-------------------------------------------------------------------------------
void valves::edit_band_out(short new_band_out)
{
   band_out = new_band_out;
}
//-------------------------------------------------------------------------------
void valves::edit_expand_norm(float new_coef)
{
   expand_norm = new_coef;
}
//-------------------------------------------------------------------------------
void valves::edit_expand_tang(float new_coef)
{
   expand_tang = new_coef;
}
//-------------------------------------------------------------------------------
void valves::set_status(short control_status)
{
   valve_status = control_status;
}
//-------------------------------------------------------------------------------
//-------------------------------------------------------------------------------
```

```
//----------------------------------------//
/    taos_display.h /
//----------------------------------------//

/ taos_display.h
/
/ This header file contains the functions used for displaying the target activated
/ offset sprayer (TAOS) control information
/
/ Written by Chris Tauzer
/ April, 1995
```

```
/
// Copyright 1995

// headers
   #include <QuickDraw.h>

// display colors
   #define MAIN_BG     0x00777777
   #define INSET_BG    0x00999999
   #define WHITE       0x00ffffff
   #define BLACK       0x00000000
   #define RED         0x00ff0000
   #define GREEN       0x0000ff00
   #define BLUE      0x000000ff
   #define DKGRAY         0x00333333
   #define MEDGRAY        0x00888888
   #define LTGRAY         0x00bbbbbb

// misc. rectangles
   Rect vd_1_maxrect, valve_rect, overspeed_rect, speed_rect, speed_disp_rect;

// program control
   Rect do_vd_edit_rect, do_valve_edit_rect, do_roi_edit_rect,
         do_show_data_rect, do_show_pixels_rect, do_image_map_rect, do_save_rect,
         shadow_rect, shadow_control_rect, pattern_rect, pattern_control_rect,
         mode_rect, mode_control_rect, radar_rect;//, simSpeedRect[11];

// overspeed rectangles
   Rect overspeed_rect0, overspeed_rect1, overspeed_rect2, overspeed_rect3;

// video digitizer adjustment rectangles
   Rect vd_adjust_rect, vd_contrast_rect, vd_bright_rect, finished_rect,
         vd_more_contrast_rect, vd_less_contrast_rect, vd_more_bright_rect,
         vd_less_bright_rect,
         vd_hue_rect, vd_more_hue_rect, vd_less_hue_rect,
         vd_sat_rect, vd_more_sat_rect, vd_less_sat_rect,
         vd_source_rect, s_video_rect, composite_rect,
         split_video_rect, interlaced_video_rect, vd_mode_rect,
         vd_adj_finished_rect, vd_use_lut_rect, apply_lut_rect;

// roi adjustment rectangles
   Rect band_rect[MAX_BAND+1], near_side_rect, far_side_rect, horz_rect, vert_rect,
         noise_rect, control_width_rect, more_control_width_rect, less_control_width_rect,
         roi_titles, band_up[MAX_BAND+1], band_down[MAX_BAND+1], roi_finished_rect,
         delete_roi_rect, add_roi_rect, image_width_rect, less_image_rect, more_image_rect;

// valve data adjustment rectangles
   Rect vedit_rect[MAX_VALVE+1], vedit_up[MAX_VALVE+1], vedit_down[MAX_VALVE+1], dist_rect,
         tof_rect, f_lead_rect, f_lag_rect, d_lead_rect, d_lag_rect, band_in_rect,
         band_out_rect, delete_valve_rect, add_valve_rect, vedit_finished_rect,
         vedit_titles, wind_norm_coef_rect, wind_tang_coef_rect;

// valve control rectangles
   Rect valve_on_rect[MAX_VALVE+1], valve_off_rect[MAX_VALVE+1], valve_auto_rect[MAX_VALVE+1],
         valve_control_rect, all_on_rect, all_auto_rect, all_off_rect;

// image map rectangles
   Rect image_map_finished_rect, image_map_hpixel_rect, image_map_vpixel_rect,
         image_map_hsetpixel_rect, image_map_vsetpixel_rect,
         image_map_hmeter_rect, image_map_vmeter_rect, next_map_set_rect,
         more_vmeter_rect, less_vmeter_rect, more_hmeter_rect, less_hmeter_rect;

// globals
   extern unsigned long valve_action_increment;
   extern short valve_count;
   extern short max_image_v;
   extern valves valve[MAX_VALVE+1];
   extern WindowPtr subWindow;

// function declarations
   void initialize_display(short band_count, short valve_count);
   void draw_main_screen(void);
   void draw_data_area(Boolean status);
   void draw_video_area(Boolean status);
   void do_overspeed_warning(char warning_type);

   void draw_over_speed_area(Boolean status);
   void draw_speed_area(Boolean status);
   void draw_vd_adjust_area(Boolean status);
   void draw_roi_adjust_area(Boolean status, short band_count, roi zone[MAX_SET+1][MAX_BAND+1]);
   void do_text_rect(Rect rect, char *string, char align);
   void draw_up_button(Rect rect, unsigned long color);
   void draw_down_button(Rect rect, unsigned long color);
   void draw_control_buttons(Rect info_rect, short width, short height);
   void draw_cell(Rect v_rect, Rect h_rect, char *string);
   void draw_valve_adjust_area(Boolean status, short valve_count, valves valve[MAX_VALVE+1]);
   void display_valve_control(short v, short new_status);
   void draw_controls(Boolean status);
   void draw_image_map_screen(Boolean status);
   void draw_depressed_button(Rect rect);
   void make_rect(Rect *rect, Point topleft, short width, short height);

   extern void my_draw_v_level(Rect level_rect, short level, short min, short max);

//-------------------------------------------------------------------------------------
// define rectangles
void initialize_display(short band_count, short valve_count)
{
   // declare variables
      short i, width, height;
      Point a, b;
      Point topleft;

   // text lines
      SetPt(&topleft, 10, 597);
      width = 390;
      height = 16;
      make_rect(&status_rect, topleft, width, height);

   // data area
      SetPt(&topleft, 670, VD_1_WINDOW_TOP + 204);
      width = band_count * (8 + 1);
      height = 32 * (8+1);
      make_rect(&data_rect, topleft, width, height);

   // video digitizer adjustment
      SetPt(&topleft, 670, VD_1_WINDOW_TOP);
      width = 160;
      height = 320;
      make_rect(&vd_adjust_rect, topleft, width, height);

      // contrast

      SetPt(&topleft, vd_adjust_rect.left + 15, vd_adjust_rect.top + 10);
      width = 10;
      height = 256;
      make_rect(&vd_contrast_rect, topleft, width, height);

      SetPt(&topleft, vd_contrast_rect.left - 10, vd_contrast_rect.bottom + 15);
      width = 10;
      height = 16;
```

```
make_rect(&vd_more_contrast_rect, topleft, width, height);

vd_less_contrast_rect.top = vd_more_contrast_rect.top;
vd_less_contrast_rect.right = vd_contrast_rect.right + 10;
vd_less_contrast_rect.left = vd_less_contrast_rect.right - 10;
vd_less_contrast_rect.bottom = vd_more_contrast_rect.bottom;

// brightness

vd_bright_rect.left = vd_contrast_rect.right + 30;
vd_bright_rect.top = vd_contrast_rect.top;
vd_bright_rect.right = vd_bright_rect.left + 10;
vd_bright_rect.bottom = vd_contrast_rect.bottom;

vd_more_bright_rect.left = vd_bright_rect.left - 10;
vd_more_bright_rect.top = vd_more_contrast_rect.top;
vd_more_bright_rect.right = vd_more_bright_rect.left + 10;
vd_more_bright_rect.bottom = vd_more_contrast_rect.bottom;

vd_less_bright_rect.top = vd_more_contrast_rect.top;
vd_less_bright_rect.right = vd_bright_rect.right + 10;
vd_less_bright_rect.left = vd_less_bright_rect.right - 10;
vd_less_bright_rect.bottom = vd_more_contrast_rect.bottom;

// hue

SetPt(&topleft, vd_bright_rect.right + 30, vd_bright_rect.top);
width = 10;
height = 256;
make_rect(&vd_hue_rect, topleft, width, height);

SetPt(&topleft, vd_hue_rect.right + 30, vd_hue_rect.top);
width = 10;
height = 256;
make_rect(&vd_sat_rect, topleft, width, height);

vd_source_rect.top = vd_adjust_rect.bottom + 20;
vd_source_rect.left = vd_adjust_rect.left;
vd_source_rect.bottom = vd_source_rect.top + 48;
vd_source_rect.right = vd_adjust_rect.right;

s_video_rect.top = vd_source_rect.top + 5;
s_video_rect.right = vd_source_rect.right - 20;
s_video_rect.bottom = vd_source_rect.top + (vd_source_rect.bottom - vd_source_rect.top)/2 - 5;
s_video_rect.left = s_video_rect.right - (s_video_rect.bottom - s_video_rect.top);

composite_rect.bottom = vd_source_rect.bottom - 5;
composite_rect.right = vd_source_rect.right - 20;
composite_rect.top = vd_source_rect.bottom - (vd_source_rect.bottom - vd_source_rect.top)/2 + 5;
composite_rect.left = s_video_rect.right - (s_video_rect.bottom - s_video_rect.top);

vd_mode_rect.top = vd_source_rect.bottom + 20;
vd_mode_rect.left = vd_source_rect.left;
vd_mode_rect.bottom = vd_mode_rect.top + 48;
vd_mode_rect.right = vd_source_rect.right;

interlaced_video_rect.top = vd_mode_rect.top + 5;
interlaced_video_rect.right = vd_mode_rect.right - 20;
interlac ed_video_rect.bottom = vd_mode_rect.top + (vd_mode_rect.bottom - vd_mode_rect.top)/2 - 5;
interlaced_video_rect.left = interlaced_video_rect.right - (interlaced_video_rect.bottom -
interlaced_video_rect.top);

split_video_rect.bottom = vd_mode_rect.bottom - 5;
split_video_rect.right = vd_mode_rect.right - 20;
split_video_rect.top = vd_mode_rect.bottom - (vd_mode_rect.bottom - vd_mode_rect.top)/2 + 5;
split_video_rect.left = interlaced_video_rect.right - (interlaced_video_rect.bottom - interlaced_video_rect.top);
```

```
    vd_use_lut_rect.left = vd_source_rect.left;
    vd_use_lut_rect.top = vd_mode_rect.bottom + 20;
    vd_use_lut_rect.right = vd_source_rect.right;
    vd_use_lut_rect.bottom = vd_use_lut_rect.top + 24;

    apply_lut_rect.top = vd_use_lut_rect.top + 5;
    apply_lut_rect.right = vd_use_lut_rect.right - 20;
    apply_lut_rect.bottom = vd_use_lut_rect.bottom - 5;
    apply_lut_rect.left = apply_lut_rect.right - (apply_lut_rect.bottom - apply_lut_rect.top);


    SetRect(&vd_adj_finished_rect, 710, vd_use_lut_rect.bottom + 20, 760, vd_use_lut_rect.bottom + 40);

// valve status rect
    valve_rect.left = status_rect.right + 20;
    valve_rect.top = data_rect.bottom + 12;
    valve_rect.bottom = valve_rect.top + 16;
    valve_rect.right = valve_rect.left + MAX_VALVE * (12);

// travel speed rect
    speed_rect.left = status_rect.left + 4;
    speed_rect.top = status_rect.top - 38;
    speed_rect.right = speed_rect.left + 200;
    speed_rect.bottom = speed_rect.top + 16;

    speed_disp_rect.left = speed_rect.right + 40;
    speed_disp_rect.top = speed_rect.top - 4;
    speed_disp_rect.right = speed_disp_rect.left + 100;
    speed_disp_rect.bottom = speed_rect.bottom + 4;

// overspeed rect
    width = 14;
    height = 14;
    a.h = speed_rect.right + 150;
    a.v = speed_rect.top - 4;
    b.h = a.h + width + 2;
    b.v = a.v + height + 2;

    SetRect(&overspeed_rect0, a.h, a.v, a.h + width, a.v + height);
    SetRect(&overspeed_rect1, b.h, a.v, b.h + width, a.v + height);
    SetRect(&overspeed_rect2, a.h, b.v, a.h + width, b.v + height);
    SetRect(&overspeed_rect3, b.h, b.v, b.h + width, b.v + height);
    SetRect(&overspeed_rect, a.h, a.v, b.h + width, b.v + height);

    SetRect(&finished_rect, 750, 560, 780, 590);

// roi adjustment
    a.h = 200;
    a.v = 150;
    width = 100;
    height = 20;
    b.v = a.v + band_count * height;

    SetRect(&near_side_rect, a.h, a.v-height, a.h + width, b.v);
    SetRect(&far_side_rect, a.h + width, a.v-height, a.h + 2 * width, b.v);
    SetRect(&horz_rect, a.h + 2 * width, a.v-height, a.h + 3 * width, b.v);
    SetRect(&vert_rect, a.h + 3 * width, a.v-height, a.h + 4 * width, b.v);
    SetRect(&noise_rect, a.h + 4 * width, a.v-height, a.h + 5 * width, b.v);

    SetRect(&control_width_rect, a.h - 100, a.v - 3 * height, a.h + width, a.v - 2 *
      height);
    SetRect(&more_control_width_rect, control_width_rect.right, control_width_rect.top,
      control_width_rect.right+height/2, control_width_rect.top + height/2);
    SetRect(&less_control_width_rect, more_control_width_rect.left,
      control_width_rect.bottom - height/2, more_control_width_rect.right,
        control_width_rect.bottom);

    SetRect(&image_width_rect, a.h - 100, a.v - 5 * height, a.h + width, a.v - 4*height);
    SetRect(&more_image_rect, image_width_rect.right, image_width_rect.top,
        image_width_rect.right+height/2, image_width_rect.top + height/2);
    SetRect(&less_image_rect, more_image_rect.left, image_width_rect.bottom - height/2,
        more_image_rect.right,image_width_rect.bottom);

    for(i=0; i<band_count; i++)
    {
        SetRect(&band_rect[i], a.h - 100, a.v + i*height, noise_rect.right,
            a.v + (i+1)*height);
        SetRect(&band_up[i], band_rect[i].left, band_rect[i].top, band_rect[i].right,
            band_rect[i].top + height/2);
        SetRect(&band_down[i], band_rect[i].left, band_rect[i].bottom - height/2 + 1,
            band_rect[i].right, band_rect[i].bottom);
    }

    SetRect(&roi_titles, a.h, a.v-height, noise_rect.right, a.v);
    SetRect(&roi_finished_rect, 400, band_rect[band_count-1].bottom + 30, 450,
        band_rect[band_count-1].bottom + 50);
    SetRect (&add_roi_rect, band_rect[0].left, roi_finished_rect.top,
        near_side_rect.left + 12, roi_finished_rect.bottom);
    SetRect(&delete_roi_rect, add_roi_rect.left, add_roi_rect.bottom + height,
        add_roi_rect.right, add_roi_rect.bottom + 2*height);

// valve adjustment
    a.h = 110;
    a.v = 150;
    width = 70;
    height = 20;
    b.v = a.v + valve_count * height;

    SetRect(&dist_rect, a.h, a.v-height, a.h + width, b.v);
    SetRect(&tof_rect, a.h + width, a.v-height, a.h + 2 * width, b.v);
    SetRect(&f_lead_rect, a.h + 2 * width, a.v-height, a.h + 3 * width, b.v);
    SetRect(&f_lag_rect, a.h + 3 * width, a.v-height, a.h + 4 * width, b.v);
    SetRect(&d_lead_rect, a.h + 4 * width, a.v-height, a.h + 5 * width, b.v);
    SetRect(&d_lag_rect, a.h + 5 * width, a.v-height, a.h + 6 * width, b.v);
    SetRect(&band_in_rect, a.h + 6 * width, a.v-height, a.h + 7 * width, b.v);
    SetRect(&band_out_rect, a.h + 7 * width, a.v-height, a.h + 8 * width, b.v);
    SetRect(&wind_norm_coef_rect, a.h + 8 * width, a.v-height, a.h + 9 * width, b.v);
    SetRect(&wind_tang_coef_rect, a.h + 9 * width, a.v-height, a.h + 10 * width, b.v);

    for(i=0; i< valve_count; i++)
    {
        SetRect(&vedit_rect[i], a.h - 100, a.v + i*height, wind_tang_coef_rect.right,
            a.v + (i+1)*height);
        SetRect(&vedit_up[i], vedit_rect[i].left, vedit_rect[i].top, vedit_rect[i].right,
            vedit_rect[i].top + height/2);
        SetRect(&vedit_down[i], vedit_rect[i].left, vedit_rect[i].bottom - height/2 + 1,
            vedit_rect[i].right, vedit_rect[i].bottom);
    }

    SetRect(& vedit_titles, a.h, a.v-height, noise_rect.right, a.v);
    SetRect(&vedit_finished_rect, 400, vedit_rect[valve_count-1].bottom + 30, 450,
        vedit_rect[valve_count-1].bottom + 50);
    SetRect(&add_valve_rect, vedit_rect[0].left, vedit_finished_rect.top,
        dist_rect.left + 12, vedit_finished_rect.bottom);
    SetRect(&delete_valve_rect, add_valve_rect.left, add_valve_rect.bottom + height,
        add_valve_rect.right, add_valve_rect.bottom + 2*height);

// valve control
    a.h = valve_rect.left;
    a.v = valve_rect.bottom + 10;
    width = 12;
    height = 12;

    for(i=0; i< MAX_VALVE; i++)
    {
        SetRect(&valve_auto_rect[i], a.h + i * width, a.v, a.h + (i+1) * width, a.v +
            height);

        SetRect(&valve_on_rect[i], a.h + i * width, valve_auto_rect[i].bottom, a.h + (i+1)
            * width, valve_auto_rect[i].bottom + height);

        SetRect(&valve_off_rect[i], a.h + i * width, valve_on_rect[i].bottom, a.h + (i+1)
            * width, valve_on_rect[i].bottom + height);
    }

    SetRect(&all_auto_rect, valve_auto_rect[MAX_VALVE-1].right + 15,
        valve_auto_rect[MAX_VALVE-1].top, valve_auto_rect[MAX_VALVE-1].right + 15 + width,
        valve_auto_rect[MAX_VALVE-1].bottom);
    SetRect(&all_on_rect, valve_on_rect[MAX_VALVE-1].right + 15,
        valve_on_rect[MAX_VALVE-1].top, valve_on_rect[MAX_VALVE-1].right + 15 + width,
        valve_on_rect[MAX_VALVE-1].bottom);
    SetRect(&all_off_rect, valve_off_rect[MAX_VALVE-1].right + 15,
        valve_off_rect[MAX_VALVE-1].top, valve_off_rect[MAX_VALVE-1].right + 15 + width,
        valve_off_rect[MAX_VALVE-1].bottom);

    SetRect(&valve_control_rect, a.h, a.v, a.h + MAX_VALVE * width + 15 + width,
        a.v + 3*height);

// program control
    a.h = all_auto_rect.right + 15;
    a.v = valve_rect.top;
    width = 40;
    height = 32;

    SetRect(&radar_rect, a.h, a.v, a.h + width, a.v + height);

    SetRect(&do_show_data_rect, a.h + 1*width + 1, a.v, a.h + 2*width + 1, a.v + height);

    SetRect(&do_show_pixels_rect, a.h + 2*width + 1, a.v,
        a.h + width + 2*width + 1, a.v + height);

    SetRect(&do_save_rect, a.h + 3*width + 1, a.v,
        a.h + width + 3*width + 1, a.v + height);

    SetRect(&do_vd_edit_rect, a.h, a.v + height + 1,
        a.h + width, a.v + height + height + 1);

    SetRect(&do_roi_edit_rect, a.h + width + 1, a.v + height + 1,
        a.h + width + width + 1, a.v + height + height + 1);

    SetRect(&do_valve_edit_rect, a.h + 2 * width + 1, a.v + height + 1,
        a.h + width + 2 * width + 1, a.v + height + height + 1);

    SetRect(&do_image_map_rect, a.h + 3 * width + 1, a.v + height + 1,
        a.h + width + 3 * width + 1, a.v + height + height + 1);

    // run mode control
    width = 120;
    height = 20;
    a.h = data_rect.left;
    a.v = 60;
    SetRect(&mode_rect, a.h, a.v, a.h + width, a.v + height);
    SetRect(&mode_control_rect, mode_rect.right - 15, mode_rect.top + 5, mode_rect.right - 5,
        mode_rect.bottom - 5);

    // shadow lut control
    a.v = mode_rect.bottom + 5;
```

```
	SetRect(&shadow_rect, a.h, a.v, a.h + width, a.v + height);
	SetRect(&shadow_control_rect, shadow_rect.right - 15, shadow_rect.top + 5, shadow_rect.right - 5,
	   shadow_rect.bottom - 5);

	// pattern control
	a.v = shadow_rect.bottom + 5;
	SetRect(&pattern_rect, a.h, a.v, a.h + width, a.v + height);
	SetRect(&pattern_control_rect, pattern_rect.right - 15, pattern_rect.top + 5,
	   pattern_rect.right - 5, pattern_rect.bottom - 5);

	// simulated speed
/*
	a.v = pattern_rect.bottom + 20;
	width = 15;
	height = 15;
	for(i = 0; i < 10; i++)
	   SetRect(&simSpeedRect[i], a.h + i*width, a.v, a.h + (i + 1)*width, a.v + height);
*/

   // image mapping
	a.h = 700;
	a.v = 60;
	width = 100;
	height = 30;

	SetRect(&next_map_set_rect, a.h, 400, a.h + 50, 420);
	SetRect(&image_map_finished_rect, a.h, 440, a.h + 50, 460);
	SetRect(&image_map_hpixel_rect, a.h, a.v, a.h + width, a.v + height);
	SetRect(&image_map_vpixel_rect, a.h, a.v + height + 10, a.h + width, a.v + 2 * height + 10);
	SetRect(&image_map_hsetpixel_rect, a.h, a.v + 4 * height, a.h + width, a.v + 5 * height);
	SetRect(&image_map_vsetpixel_rect, a.h, a.v + 5 * height + 10, a.h + width, a.v + 6 * height + 10);
	SetRect(&image_map_hmeter_rect, a.h, a.v + 7 * height, a.h + width, a.v + 8 * height);
	SetRect(&image_map_vmeter_rect, a.h, a.v + 8 * height + 10, a.h + width, a.v + 9 * height + 10);
	SetRect(&more_vmeter_rect, image_map_vmeter_rect.left - height/2 - 8, image_map_vmeter_rect.top,
	   image_map_vmeter_rect.left - 8, image_map_vmeter_rect.top + height/2);
	SetRect(&less_vmeter_rect, image_map_vmeter_rect.left - height/2 - 8, image_map_vmeter_rect.top +
height/2,
	   image_map_vmeter_rect.left - 8, image_map_vmeter_rect.bottom);
	SetRect(&more_hmeter_rect, image_map_hmeter_rect.left - height/2 - 8, image_map_hmeter_rect.top,
	   image_map_hmeter_rect.left - 8, image_map_hmeter_rect.top + height/2);
	SetRect(&less_hmeter_rect, image_map_hmeter_rect.left - height/2 - 8, image_map_hmeter_rect.top +
height/2,
	   image_map_hmeter_rect.left - 8, image_map_hmeter_rect.bottom);
}
//-----------------------------------------------------------------------------
void draw_main_screen(void)
{
	Rect fr;
	char fw = 2;

	HideCursor();

	fr.top = WINDOW_TOP; fr.left = WINDOW_LEFT; fr.bottom = 620; fr.right = 830;
	my_fill_rect(&fr, MAIN_BG, 1);
	my_draw_frame(&status_rect, fw);

	ShowCursor();
}
//-----------------------------------------------------------------------------
void draw_controls(Boolean status)
{
   char fw = 3;
   short i;

   HideCursor();

	my_fill_rect(&radar_rect, MEDGRAY, 1);
	my_draw_frame_in(&radar_rect, fw);
	my_draw_rect(radar_rect, BLACK);
	do_text_rect(radar_rect, "RDR", 1);

	my_fill_rect(&do_show_data_rect, MEDGRAY, 1);
	my_draw_frame_in(&do_show_data_rect, fw);
	my_draw_rect(do_show_data_rect, BLACK);
	do_text_rect(do_show_data_rect, "DAT", 1);

	my_fill_rect(&do_show_pixels_rect, MEDGRAY, 1);
	my_draw_frame_in(&do_show_pixels_rect, fw);
	my_draw_rect(do_show_pixels_rect, BLACK);
	do_text_rect(do_show_pixels_rect, "PIX", 1);

	my_fill_rect(&do_save_rect, MEDGRAY, 1);
	my_draw_frame_in(&do_save_rect, fw);
	my_draw_rect(do_save_rect, BLACK);
	do_text_rect(do_save_rect, " SAVE", 0);

	my_fill_rect(&do_vd_edit_rect, MEDGRAY, 1);
	my_draw_frame_in(&do_vd_edit_rect, fw);
	my_draw_rect(do_vd_edit_rect, BLACK);
	do_text_rect(do_vd_edit_rect, "VD", 1);

	my_fill_rect(&do_roi_edit_rect, MEDGRAY, 1);
	my_draw_frame_in(&do_roi_edit_rect, fw);
	my_draw_rect(do_roi_edit_rect, BLACK);
	do_text_rect(do_roi_edit_rect, "ROI", 1);

	my_fill_rect(&do_valve_edit_rect, MEDGRAY, 1);
	my_draw_frame_in(&do_valve_edit_rect, fw);
	my_draw_rect(do_valve_edit_rect, BLACK);
	do_text_rect(do_valve_edit_rect, "VAL", 1);

	my_fill_rect(&do_image_map_rect, MEDGRAY, 1);
	my_draw_frame_in(&do_image_map_rect, fw);
	my_draw_rect(do_image_map_rect, BLACK);
	do_text_rect(do_image_map_rect, " MAP", 0);

	fw = 2;

	// run mode
	my_fill_rect(&mode_rect, INSET_BG, 1);
	my_draw_rect(mode_rect, BLACK);
	do_text_rect(mode_rect, " AUTO", 0);
	my_draw_rect(mode_control_rect, BLACK);
	my_fill_rect(&mode_control_rect, WHITE, 1);

	// shadow lut
	my_fill_rect(&shadow_rect, INSET_BG, 1);
	my_draw_rect(shadow_rect, BLACK);
	do_text_rect(shadow_rect, " SHADOWS", 0);
	my_draw_rect(shadow_control_rect, BLACK);
	my_fill_rect(&shadow_control_rect, WHITE, 1);

	// expand pattern
	my_fill_rect(&pattern_rect, INSET_BG, 1);
	my_draw_rect(pattern_rect, BLACK);
	do_text_rect(pattern_rect, " PATTERN", 0);
	my_draw_rect(pattern_control_rect, BLACK);
	my_fill_rect(&pattern_control_rect, WHITE, 1);

	// simulated speed
/*
	for(i=0; i<10; i++)
```

```
    {
      my_fill_rect(&simSpeedRect[i], INSET_BG, 1);
      my_draw_rect(simSpeedRect[i], BLACK);
    }
*/
  ShowCursor();

}
//-----------------------------------------------------------------------------------------------
void draw_data_area(Boolean status)
{
      Rect fr;
      short i;
      char fw = 2;

  HideCursor();

  if(status)
  {
     // data plant info rectangle
        my_fill_rect(&data_rect, INSET_BG, 1);
        my_draw_frame(&data_rect, fw);

    // valve control display rectangle
        my_fill_rect(&valve_rect, INSET_BG, 1);
        my_draw_frame(&valve_rect, fw);

        SetRect(&fr, valve_control_rect.left, valve_control_rect.top,
           valve_auto_rect[MAX_VALVE-1].right, valve_control_rect.bottom);
        my_fill_rect(&fr, INSET_BG, 1);
        my_draw_frame(&fr, fw);

        for(i = 0; i < MAX_VALVE; i++)
        {
           display_valve_control(i, valve[i].get_status());
        }

        my_fill_rect(&all_auto_rect, MEDGRAY, 1);
        my_draw_frame_in(&all_auto_rect, 2);
        my_draw_rect(all_auto_rect, BLACK);

        my_fill_rect(&all_on_rect, MEDGRAY, 1);
        my_draw_frame_in(&all_on_rect, 2);
        my_draw_rect(all_on_rect, BLACK);

        my_fill_rect(&all_off_rect, MEDGRAY, 1);
        my_draw_frame_in(&all_off_rect, 2);
        my_draw_rect(all_off_rect, BLACK);

  }
  else
  {
     fw += 2;
     fr.left = data_rect.left-fw;
     fr.top = data_rect.top-fw;
     if(valve_rect.right >= data_rect.right)
        fr.right = valve_rect.right+fw;
     else
        fr.right = data_rect.right+fw;
     fr.bottom = data_rect.top + 9*32 + fw;
     my_fill_rect(&fr, MAIN_BG, 1);
  }

  ShowCursor();
}
//-----------------------------------------------------------------------------------------------
```

```
void display_valve_control(short v, short new_status)
{
  Rect rect;

  HideCursor();

  switch(new_status)
  {
    case AUTO:
       my_fill_rect(&valve_auto_rect[v], MEDGRAY, 1);
       my_draw_frame_in(&valve_auto_rect[v], 2);
       my_draw_rect(valve_auto_rect[v], BLACK);
       my_fill_rect(&valve_on_rect[v], DKGRAY, 1);
       my_fill_rect(&valve_off_rect[v], DKGRAY, 1);
       SetRect(&rect, valve_on_rect[v].left, valve_auto_rect[v].top,
          valve_on_rect[v].right, valve_off_rect[v].bottom);
       my_draw_rect(rect, BLACK);
       break;
    case MANUAL_ON:
       my_fill_rect(&valve_auto_rect[v], DKGRAY, 1);
       my_fill_rect(&valve_on_rect[v], MEDGRAY, 1);
       my_draw_frame_in(&valve_on_rect[v], 2);
       my_draw_rect(valve_on_rect[v], BLACK);
       my_fill_rect(&valve_off_rect[v], DKGRAY, 1);
       SetRect(&rect, valve_on_rect[v].left, valve_auto_rect[v].top,
          valve_on_rect[v].right, valve_off_rect[v].bottom);
       my_draw_rect(rect, BLACK);
       break;
    case MANUAL_OFF:
       my_fill_rect(&valve_auto_rect[v], DKGRAY, 1);
       my_fill_rect(&valve_on_rect[v], DKGRAY, 1);
       my_fill_rect(&valve_off_rect[v], MEDGRAY, 1);
       my_draw_frame_in(&valve_off_rect[v], 2);
       my_draw_rect(valve_off_rect[v], BLACK);
       SetRect(&rect, valve_on_rect[v].left, valve_auto_rect[v].top,
          valve_on_rect[v].right, valve_off_rect[v].bottom);
       my_draw_rect(rect, BLACK);
  }
  ShowCursor();
}

//-----------------------------------------------------------------------------------------------
void draw_roi_adjust_area(Boolean status, short band_count, roi zone[MAX_SET+1][MAX_BAND+1])
{
  Rect fr;
  char fw = 2;
  short i;
  char string[50] = {0};
  Rect info_rect;
  short width = 100;
  short height = 20;
  Point a, b;

  a.h = 200;
  a.v = 150;
  b.v = a.v + band_count * height;

  HideCursor();

  // frame
    SetRect(&fr, band_rect[0].left, band_rect[0].top, band_rect[0].right,
       band_rect[band_count-1].bottom);
    my_draw_frame(&fr, fw);
    SetRect(&fr, near_side_rect.left, near_side_rect.top, noise_rect.right,
       band_rect[band_count-1].bottom);
    my_draw_frame(&fr, fw);
```

```
// re-define bottom of table (adjustment for adding/deleting bands)
   b.v = a.v + band_count * height;
   band_rect[band_count - 1].bottom = b.v;
   near_side_rect.bottom = b.v;
   far_side_rect.bottom = b.v;
   horz_rect.bottom = b.v;
   vert_rect.bottom = b.v;
   noise_rect.bottom = b.v;

   SetRect(&roi_finished_rect, 400, band_rect[band_count-1].bottom + 30, 450,
      band_rect[band_count-1].bottom + 50);
   SetRect(&add_roi_rect, band_rect[0].left, roi_finished_rect.top,
      near_side_rect.left + 12, roi_finished_rect.bottom);
   SetRect(&delete_roi_rect, add_roi_rect.left, add_roi_rect.bottom + height,
      add_roi_rect.right, add_roi_rect.bottom + 2*height);

// draw band rectangles
   for(i = 0; i < band_count; i++)
   {
      my_fill_rect(&band_rect[i], WHITE, 1);
      my_draw_rect(band_rect[i], BLACK);
      sprintf(string, " Band %i:", i+1);
      do_text_rect(band_rect[i], string, 0);
   }

// draw value grid rectangles
   my_fill_rect(&near_side_rect, WHITE, 1);
   my_draw_rect(near_side_rect, BLACK);
   my_fill_rect(&far_side_rect, WHITE, 1);
   my_draw_rect(far_side_rect,BLACK);
   my_fill_rect(&horz_rect, WHITE, 1);
   my_draw_rect(horz_rect, BLACK);
   my_fill_rect(&vert_rect, WHITE, 1);
   my_draw_rect(vert_rect,BLACK);
   my_fill_rect(&noise_rect, WHITE, 1);
   my_draw_rect(noise_rect, BLACK);

// draw control width rectangle
   my_fill_rect(&control_width_rect, WHITE, 1);
   my_draw_rect(control_width_rect, BLACK);
   SetRect(&fr, control_width_rect.left, control_width_rect.top,
      more_control_width_rect.right, control_width_rect.bottom);
   my_draw_frame(&fr, fw);
   sprintf(string, " Control Width: %3.2f m", (float)valve_action_increment / RAVENS);
   do_text_rect(control_width_rect, string, 0);
   draw_up_button(more_control_width_rect, WHITE);
   draw_down_button(less_control_width_rect, WHITE);

// draw image width rectangle
   my_fill_rect(&image_width_rect, WHITE, 1);
   my_draw_rect(image_width_rect, BLACK);
   SetRect(&fr, image_width_rect.left, image_width_rect.top, more_image_rect.right,
      image_width_rect.bottom);
   my_draw_frame(&fr, fw);
   sprintf(string, " Image Width: %i pixels", max_image_v);
   do_text_rect(image_width_rect, string, 0);
   draw_up_button(more_image_rect, WHITE);
   draw_down_button(less_image_rect, WHITE);

// fill in values
   for(i=0; i< band_count; i++)
   {
      sprintf(string, "%3.2f", zone[0][i].roi_nearside());
      draw_cell(near_side_rect, band_rect[i], string);

      sprintf(string, "%3.2f", zone[0][i].roi_farside());
      draw_cell(far_side_rect, band_rect[i], string);

      sprintf(string, "%i", zone[0][i].roi_horz_skip());
      draw_cell(horz_rect, band_rect[i], string);

      sprintf(string, "%i", zone[0][i].roi_vert_skip());
      draw_cell(vert_rect, band_rect[i], string);

      sprintf(string, "%i", zone[0][i].roi_noise());
      draw_cell(noise_rect, band_rect[i], string);
   }

// write out titles
   SetRect(&info_rect, near_side_rect.left, roi_titles.top, near_side_rect.right,
      roi_titles.bottom);
   do_text_rect(info_rect, " Nearside, m", 0);
   SetRect(&info_rect, far_side_rect.left, roi_titles.top, far_side_rect.right,
      roi_titles.bottom);
   do_text_rect(info_rect, " Farside, m", 0);
   SetRect(&info_rect, horz_rect.left, roi_titles.top, horz_rect.right,
      roi_titles.bottom);
   do_text_rect(info_rect, "Horz_Incr, pxl", 0);
   SetRect(&info_rect, vert_rect.left, roi_titles.top, vert_rect.right,
      roi_titles.bottom);
   do_text_rect(info_rect, "Vert_Incr, pxl", 0);
   SetRect(&info_rect, noise_rect.left, roi_titles.top, noise_rect.right,
      roi_titles.bottom);
   do_text_rect(info_rect, " Noise, pxl", 0);

// add or delete band
   my_fill_rect(&add_roi_rect, WHITE, 1);
   my_draw_frame(&add_roi_rect, fw);
   do_text_rect(add_roi_rect, " ADD BAND", 1);
   my_fill_rect(&delete_roi_rect, WHITE, 1);
   my_draw_frame(&delete_roi_rect, fw);
   sprintf(string, "DELETE BAND: %i", band_count);
   do_text_rect(delete_roi_rect, string, 0);

// draw finished rectangle
   my_fill_rect(&roi_finished_rect, WHITE, 1);
   my_draw_frame(&roi_finished_rect, fw);
   do_text_rect(roi_finished_rect, " DONE", 1);

   ShowCursor();

}
//------------------------------------------------------------------------------------------
void draw_valve_adjust_area(Boolean status, short valve_count, valves valve[MAX_VALVE])
{
   Rect fr;
   char fw = 2;
   short i;
   char string[50] = {0};
   Rect info_rect;
   short width = 100;
   short height = 20;
   Point a, b;


   a.h = 200;
   a.v = 150;
   b.v = a.v + valve_count * height;

   HideCursor();
```

```
// frame
   SetRect(&fr, vedit_rect[0].left, vedit_rect[0].top, vedit_rect[0].right,
      vedit_rect[valve_count-1].bottom);
   my_draw_frame(&fr, fw);
   SetRect(&fr, dist_rect.left, dist_rect.top, wind_tang_coef_rect.right,
      vedit_rect[valve_count-1].bottom);
   my_draw_frame(&fr, fw);

// re-define bottom of table (adjustment for adding/deleting valves)
   b.v = a.v + valve_count * height;
   vedit_rect[valve_count - 1].bottom = b.v;
   dist_rect.bottom = b.v;
   tof_rect.bottom = b.v;
   f_lead_rect.bottom = b.v;
   f_lag_rect.bottom = b.v;
   d_lead_rect.bottom = b.v;
   d_lag_rect.bottom = b.v;
   band_in_rect.bottom = b.v;
   band_out_rect.bottom = b.v;
   wind_norm_coef_rect.bottom = b.v;
   wind_tang_coef_rect.bottom = b.v;

   SetRect(&vedit_finished_rect, 400, vedit_rect[valve_count-1].bottom + 30, 450,
      vedit_rect[valve_count-1].bottom + 50);

// draw vedit rectangles
   for(i = 0; i < valve_count; i++)
   {
      my_fill_rect(&vedit_rect[i], WHITE, 1);
      my_draw_rect(vedit_rect[i], BLACK);
      sprintf(string, " Valve %i:", i+1);
      do_text_rect(vedit_rect[i], string, 0);
   }

// draw value grid rectangles
   my_fill_rect(&dist_rect, WHITE, 1);
   my_draw_rect(dist_rect, BLACK);
   my_fill_rect(&tof_rect, WHITE, 1);
   my_draw_rect(tof_rect,BLACK);
   my_fill_rect(&f_lead_rect, WHITE, 1);
   my_draw_rect(f_lead_rect, BLACK);
   my_fill_rect(&f_lag_rect, WHITE, 1);
   my_draw_rect(f_lag_rect,BLACK);
   my_fill_rect(&d_lead_rect, WHITE, 1);
   my_draw_rect(d_lead_rect, BLACK);
   my_fill_rect(&d_lag_rect, WHITE, 1);
   my_draw_rect(d_lag_rect,BLACK);
   my_fill_rect(&band_in_rect, WHITE, 1);
   my_draw_rect(band_in_rect,BLACK);
   my_fill_rect(&band_out_rect, WHITE, 1);
   my_draw_rect(band_out_rect,BLACK);
   my_fill_rect(&wind_norm_coef_rect, WHITE, 1);
   my_draw_rect(wind_norm_coef_rect,BLACK);
   my_fill_rect(&wind_tang_coef_rect, WHITE, 1);
   my_draw_rect(wind_tang_coef_rect,BLACK);

// fill in values
   for(i=0; i< valve_count; i++)
   {
      sprintf(string, "%4.3f", valve[i].get_dist());
      draw_cell(dist_rect, vedit_rect[i], string);
      sprintf(string, "%4.3f", valve[i].get_tof());
      draw_cell(tof_rect, vedit_rect[i], string);
      sprintf(string, "%4.3f", valve[i].get_f_lead());
      draw_cell(f_lead_rect, vedit_rect[i], string);
      sprintf(string, "%4.3f", valve[i].get_f_lag());
      draw_cell(f_lag_rect, vedit_rect[i], string);
      sprintf(string, "%4.3f", valve[i].get_d_lead());
      draw_cell(d_lead_rect, vedit_rect[i], string);
      sprintf(string, "%4.3f", valve[i].get_d_lag());
      draw_cell(d_lag_rect, vedit_rect[i], string);
      sprintf(string, "%i", valve[i].get_band_in());
      draw_cell(band_in_rect, vedit_rect[i], string);
      sprintf(string, "%i", valve[i].get_band_out());
      draw_cell(band_out_rect, vedit_rect[i], string);
      sprintf(string, "%3.1f", valve[i].get_expand_norm());
      draw_cell(wind_norm_coef_rect, vedit_rect[i], string);
      sprintf(string, "%3.1f", valve[i].get_expand_tang());
      draw_cell(wind_tang_coef_rect, vedit_rect[i], string);
   }

// write out titles
   SetRect(&info_rect, dist_rect.left, vedit_titles.top, dist_rect.right,
      vedit_titles.bottom);
   do_text_rect(info_rect, " dist, m", 0);
   SetRect(&info_rect, tof_rect.left, vedit_titles.top, tof_rect.right,
      vedit_titles.bottom);
   do_text_rect(info_rect, " tof adj)", 0);
   SetRect(&info_rect, f_lead_rect.left, vedit_titles.top, f_lead_rect.right,
      vedit_titles.bottom);
   do_text_rect(info_rect, "lead (f), m", 0);
   SetRect(&info_rect, f_lag_rect.left, vedit_titles.top, f_lag_rect.right,
      vedit_titles.bottom);
   do_text_rect(info_rect, "lag (f), m", 0);
   SetRect(&info_rect, d_lead_rect.left, vedit_titles.top, d_lead_rect.right,
      vedit_titles.bottom);
   do_text_rect(info_rect, " lead (d), m", 0);
   SetRect(&info_rect, d_lag_rect.left, vedit_titles.top, d_lag_rect.right,
      vedit_titles.bottom);
   do_text_rect(info_rect, " lag (d), m", 0);
   SetRect(&info_rect, band_in_rect.left, vedit_titles.top, band_in_rect.right,
      vedit_titles.bottom);
   do_text_rect(info_rect, " in band", 0);
   SetRect(&info_rect, band_out_rect.left, vedit_titles.top, band_out_rect.right,
      vedit_titles.bottom);
   do_text_rect(info_rect, " out band", 0);
   SetRect(&info_rect, wind_norm_coef_rect.left, vedit_titles.top, wind_norm_coef_rect.right,
      vedit_titles.bottom);
   do_text_rect(info_rect, " n expand", 0);
   SetRect(&info_rect, wind_tang_coef_rect.left, vedit_titles.top, wind_tang_coef_rect.right,
      vedit_titles.bottom);
   do_text_rect(info_rect, " t expand", 0);

// draw finished rectangle
   my_fill_rect(&vedit_finished_rect, WHITE, 1);
   my_draw_frame(&vedit_finished_rect, fw);
   do_text_rect(vedit_finished_rect, " DONE", 1);

   ShowCursor();

}
//---------------------------------------------------------------------------------
// draw screen for adjusting image map
void draw_image_map_screen(Boolean status)
{
   short fw = 2;

   HideCursor();

      draw_main_screen();
```

```
 // draw finished rectangle

     my_fill_rect(&next_map_set_rect, WHITE, 1);
     my_draw_frame(&next_map_set_rect, fw);
     do_text_rect(next_map_set_rect, " NEXT", 1);

     my_fill_rect(&image_map_finished_rect, WHITE, 1);
     my_draw_frame(&image_map_finished_rect, fw);
     do_text_rect(image_map_finished_rect, " DONE", 1);

     my_fill_rect(&image_map_hpixel_rect, WHITE, 1);
     my_draw_frame(&image_map_hpixel_rect, fw);

     my_fill_rect(&image_map_vpixel_rect, WHITE, 1);
     my_draw_frame(&image_map_vpixel_rect, fw);

     my_fill_rect(&image_map_hsetpixel_rect, WHITE, 1);
     my_draw_frame(&image_map_hsetpixel_rect, fw);

     my_fill_rect(&image_map_vsetpixel_rect, WHITE, 1);
     my_draw_frame(&image_map_vsetpixel_rect, fw);

     my_fill_rect(&image_map_hmeter_rect, WHITE, 1);
     my_draw_frame(&image_map_hmeter_rect, fw);

     draw_up_button(more_hmeter_rect, INSET_BG);
     draw_down_button(less_hmeter_rect, INSET_BG);


     my_fill_rect(&image_map_vmeter_rect, WHITE, 1);
     my_draw_frame(&image_map_vmeter_rect, fw);

     draw_up_button(more_vmeter_rect, INSET_BG);
     draw_down_button(less_vmeter_rect, INSET_BG);


   ShowCursor();

}
//---------------------------------------------------------------------------------
void make_rect(Rect *rect, Point topleft, short width, short height)
{
   SetRect(rect, topleft.h, topleft.v, topleft.h + width, topleft.v + height);
}
//---------------------------------------------------------------------------------
// write text in table cell
void draw_cell(Rect h_rect, Rect v_rect, char *string)
{
   Rect data_rect;

   SetRect(&data_rect, h_rect.left, v_rect.top, h_rect.right,
      v_rect.bottom);
   my_draw_rect(data_rect, BLACK);
   do_text_rect(data_rect, string, 1);
   draw_control_buttons(data_rect, 10, 10);
}
//---------------------------------------------------------------------------------
void draw_vd_adjust_area(Boolean status)
{
     Rect fr;
     char fw = 2;

   HideCursor();

   if(status)
   {
```

```
// vd adjust rectangle
   my_fill_rect(&vd_adjust_rect, INSET_BG, 1);
   my_draw_frame(&vd_adjust_rect, fw);

   my_fill_rect(&vd_contrast_rect, INSET_BG, 1);
   my_draw_frame(&vd_contrast_rect, 2);

   draw_up_button(vd_more_contrast_rect, INSET_BG);
   my_draw_frame(&vd_more_contrast_rect, 2);

   draw_down_button(vd_less_contrast_rect, INSET_BG);
   my_draw_frame(&vd_less_contrast_rect, 2);

   my_fill_rect(&vd_bright_rect, INSET_BG, 1);
   my_draw_frame(&vd_bright_rect, 2);

   draw_up_button(vd_more_bright_rect, INSET_BG);
   my_draw_frame(&vd_more_bright_rect, 2);

   draw_down_button(vd_less_bright_rect, INSET_BG);
   my_draw_frame(&vd_less_bright_rect, 2);

   my_fill_rect(&vd_hue_rect, INSET_BG, 1);
   my_draw_frame(&vd_hue_rect, 2);

   my_fill_rect(&vd_sat_rect, INSET_BG, 1);
   my_draw_frame(&vd_sat_rect, 2);

   my_fill_rect(&vd_source_rect, INSET_BG, 1);
   my_draw_frame(&vd_source_rect, 2);

   fr.left = vd_source_rect.left;
   fr.top = vd_source_rect.top;
   fr.right = vd_source_rect.right;
   fr.bottom = vd_source_rect.top + (vd_source_rect.bottom - vd_source_rect.top)/2.0;
   do_text_rect(fr, " S-Video", 0);

   fr.left = vd_source_rect.left;
   fr.bottom = vd_source_rect.bottom;
   fr.right = vd_source_rect.right;
   fr.top = vd_source_rect.top + (vd_source_rect.bottom - vd_source_rect.top)/2.0;
   do_text_rect(fr, " Composite", 0);

   my_fill_rect(&vd_mode_rect, INSET_BG, 1);
   my_draw_frame(&vd_mode_rect, 2);

   fr.left = vd_mode_rect.left;
   fr.top = vd_mode_rect.top;
   fr.right = vd_mode_rect.right;
   fr.bottom = vd_mode_rect.top + (vd_mode_rect.bottom - vd_mode_rect.top)/2.0;
   do_text_rect(fr, " Interlace", 0);

   fr.left = vd_mode_rect.left;
   fr.bottom = vd_mode_rect.bottom;
   fr.right = vd_mode_rect.right;
   fr.top = vd_mode_rect.top + (vd_mode_rect.bottom - vd_mode_rect.top)/2.0;
   do_text_rect(fr, " Even / Odd", 0);

   my_fill_rect(&vd_use_lut_rect, INSET_BG, 1);
   my_draw_frame(&vd_use_lut_rect, 2);

   fr.left = vd_use_lut_rect.left;
   fr.bottom = vd_use_lut_rect.bottom;
   fr.right = vd_use_lut_rect.right;
   fr.top = vd_use_lut_rect.top;
   do_text_rect(fr, " Apply LUT", 0);
```

```
        my_fill_rect(&apply_lut_rect, WHITE, 1);
        my_draw_rect(apply_lut_rect, BLACK);

        my_fill_re ct(&s_video_rect, WHITE, 1);
        my_draw_rect(s_video_rect, BLACK);

        my_fill_rect(&composite_rect, WHITE, 1);
        my_draw_rect(composite_rect, BLACK);

        my_fill_rect(&split_video_rect, WHITE, 1);
        my_draw_rect(split_video_rect, BLACK);

        my_fill_rect(&interlaced_video_rect, WHITE, 1);
        my_draw_rect(interlaced_video_rect, BLACK);

        my_fill_rect(&vd_adj_finished_rect, INSET_BG, 1);
        my_draw_frame(&vd_adj_finished_rect, fw);

        do_text_rect(vd_adj_finished_rect, "  DONE", 0);
    }
    else
    {
      fw += 2;
      fr.left = vd_adjust_rect.left-fw;
      fr.top = vd_adjust_rect.top-fw;
      fr.right = vd_adjust_rect.right+fw;
      fr.bottom = vd_adj_finished_rect.bottom+fw;
      my_fill_rect(&fr, MAIN_BG, 1);

    }

    ShowCursor();
}
//-------------------------------------------------------------------------------------
void draw_video_area(Boolean status)
{
      Rect fr;
      char fw = 4;

    HideCursor();

    if(status)
    {
    // video display rectangle
      fr.top = vd[0].image_rect.top + VD_1_WINDOW_TOP - 8;
      fr.left = vd[0].image_rect.left + VD_1_WINDOW_LEFT - 4;
      fr.bottom = vd[0].image_rect.bottom + VD_1_WINDOW_TOP - 8;
      fr.right = vd[0].image_rect.right + VD_1_WINDOW_LEFT - 4;
      my_fill_rect(&fr, INSET_BG, 1);
      my_draw_frame(&fr, fw);
    }
    else
    {
      fw += 2;
      fr.left = vd[0].image_rect.left + VD_1_WINDOW_LEFT - 4 - fw;
      fr.top = vd[0].image_rect.top + VD_1_WINDOW_TOP - 8 - fw;
      fr.right = vd[0].image_rect.right + VD_1_WINDOW_LEFT - 4 + fw;
      fr.bottom = vd[0].image_rect.bottom + VD_1_WINDOW_TOP - 8 + fw;
      my_fill_rect(&fr, MAIN_BG, 1);
    }

    ShowCursor();
}
//-------------------------------------------------------------------------------------
void draw_over_speed_area(Boolean status)
{
    Rect fr;
    char fw = 2;

    HideCursor();

    if(status)
    {
      my_draw_frame(&overspeed_rect, fw);
      my_fill_rect(&overspeed_rect0, 0x0000000, 1);
      my_fill_rect(&overspeed_rect1, 0x0000000, 1);
      my_fill_rect(&overspeed_rect2, 0x0000000, 1);
      my_fill_rect(&overspeed_rect3, 0x0000000, 1);
    }
    else
    {
      fw += 2;
      fr.left = overspeed_rect.left-fw;
      fr.top = overspeed_rect.top-fw;
      fr.right = overspeed_rect.right+fw;
      fr.bottom = overspeed_rect.bottom+fw;
      my_fill_rect(&fr, MAIN_BG, 1);
    }

    ShowCursor();
}

//-------------------------------------------------------------------------------------
void draw_speed_area(Boolean status)
{
      Rect fr;
      char fw = 4;

    HideCursor();

    if(status)
    {
      fr.top = speed_rect.top - (fw + 10);
      fr.left = speed_rect.left - (fw + 1);
      fr.bottom = speed_rect.bottom + (fw + 10);
      fr.right = speed_rect.right + (fw + 1);
      my_fill_rect(&fr, INSET_BG, 1);
      my_draw_frame(&fr, 2);
      my_fill_rect(&speed_rect, INSET_BG, 1);
      my_draw_frame(&speed_rect,fw);

      my_fill_rect(&speed_disp_rect, WHITE, 1);
      my_draw_frame(&speed_disp_rect,fw);
    }
    else
    {
      fw += 14;
      fr.left = speed_rect.left-fw;
      fr.top = speed_rect.top-fw;
      fr.right =speed_rect.right+fw;
      fr.bottom = speed_rect.bottom+fw;
      my_fill_rect(&fr, MAIN_BG, 1);
    }

    ShowCursor();
}
//-------------------------------------------------------------------------------------
void do_text_rect(Rect rect, char *string, char align)
{
    short v_center, h_center;
    size_t length;
```

```
  char print_string[100] = {0};

  sprintf(print_string, "%s", string);

  v_center = rect.top + (rect.bottom - rect.top)/2;

  if(align == 1)
  {
     h_center = rect.left + (rect.right - rect.left)/2 - 1;
     length = strlen(string);
     MoveTo(h_center - ((length/2) * 10), v_center + 4);
  }
  else
     MoveTo(rect.left + 2, v_center + 4);

  drawstring(print_string);

}
//------------------------------------------------------------------------------------------
void draw_up_button(Rect rect, unsigned long color)
{
  short x, y, x1, x2;
  short mid = rect.left + my_round(0.5 * (rect.right - rect.left));
  short ystart = rect.top + (rect.bottom - rect.top ) / 4;
  short yend = rect.bottom - (rect.bottom - rect.top ) / 4;

  x1 = x2 = mid;

  HideCursor();

  my_fill_rect(&rect, color, 1);
  my_draw_rect(rect, BLACK);

  for(y = ystart; y < yend-1; y++)
  {
     for(x = x1; x < x2; x++)
        my_draw_point(x, y, BLACK);
     x1--;
     x2++;
  }

  my_draw_frame_in(&rect, 2);
  my_draw_rect(rect, BLACK);

  ShowCursor();
}
//------------------------------------------------------------------------------------------
void draw_down_button(Rect rect, unsigned long color)
{
  short x, y, x1, x2;
  short mid = rect.left + my_round(0.5 * (rect.right - rect.left));
  short yend = rect.top + my_round((rect.bottom - rect.top ) / 4);
  short ystart = rect.bottom - my_round((rect.bottom - rect.top ) / 4);

  HideCursor();

  x1 = x2 = mid;

  my_fill_rect(&rect, color, 1);

  for(y = ystart; y > yend+2; y--)
  {
     for(x = x1; x <= x2; x++)
        my_draw_point(x, y, BLACK);
     x1--;
     x2++;
  }

  my_draw_frame_in(&rect, 2);
  my_draw_rect(rect, BLACK);

  ShowCursor();

}
//------------------------------------------------------------------------------------------
void draw_depressed_button(Rect rect)
{
  short i, width = 2;

  HideCursor();
  for(i = 0; i < width; i++)
  {
     my_draw_rect(rect, BLACK);
     rect.left += 1;
     rect.top += 1;
     rect.right -= 1;
     rect.bottom -= 1;
  }
  my_draw_rect(rect, DKGRAY);
  ShowCursor();
}
//------------------------------------------------------------------------------------------
void draw_control_buttons(Rect info_rect, short width, short height)
{
     Rect u_button_rect, d_button_rect;

     SetRect(&u_button_rect, info_rect.right - width, info_rect.top, info_rect.right,
          info_rect.top+height);
     draw_up_button(u_button_rect, WHITE);
     SetRect(&d_button_rect, u_button_rect.left, u_button_rect.bottom,
        u_button_rect.right, info_rect.bottom);
     draw_down_button(d_button_rect, WHITE);
}
//------------------------------------------------------------------------------------------
// ----- warn if excessive speed (valve control, frame grab, control loop) -----

void do_overspeed_warning(char warning_type)
{
  Rect *warning_rect;
  unsigned long color;
  char skip;
  static short last_status0=0, last_status1=0, last_status2=0, last_status3=0;
  short *status;
  unsigned char din_c;
  char warning[4];

  // do overspeed warning at operator console
     read_dio(C, &din_c);
     write_dio(C, din_c | OVER_SPEED);
}

/*
  // select location and color for type of overspeed warning on monitor
     switch(warning_type)
     {
        case(0):
           last_status0 = (last_status0) ? 0: 1;
           *status = last_status0;
           warning_rect = &overspeed_rect0;
           color = 0x00ff0000;
           skip = 1;
           sprintf(warning, "V");
          break;
        case(1):
          last_status1 = (last_status1) ? 0: 1;
          *status = last_status1;
          warning_rect = &overspeed_rect1;
          color = 0x0000ff00;
          skip = 1;
          sprintf(warning, "I");
          break;
        case(2):
          last_status2 = (last_status2) ? 0: 1;
          *status = last_status2;
          warning_rect = &overspeed_rect2;
          color = 0x000000ff;
          skip = 1;
          sprintf(warning, "P");
          break;
        case(3):
          last_status3 = (last_status3) ? 0: 1;
          *status = last_status3;
          warning_rect = &overspeed_rect3;
          color = 0x00ffff00;
          skip = 1;
          sprintf(warning, "O");
          break;
      }

   // do warning on monitor
      if(*status == 0)
        my_fill_rect(warning_rect, color, skip);
      else
        my_fill_rect(warning_rect, 0x00000000, skip);

      do_text_rect(*warning_rect, warning, 1);

//-----------------------------------------//
/    taos_interface.h  /
//-----------------------------------------//

// -----------------------------------------------------------------------------
/
/ taos_interface.h
/
// written by Chris Tauzer
/ 1995
/
// This file contains the functions used for controlling the TAOS system using computer
/ input.
/
// -----------------------------------------------------------------------------

EventRecord     gEvent;
short           gwindowCode;
Boolean         guserDone;
CursHandle      gtheCursor;
WindowPtr       gwhichWindow;

extern WindowPtr subWindow, mainWindow;
/////extern unsigned long firstTick, last_count;
/////extern unsigned long overflow_record;

/////extern short simSpeed;

void EventTask();
void DoMouseDown(EventRecord *event);
void buttons(Point where);
short mouse_in(Rect c_rect, Point click);
void DoEvent(void);
Boolean Do_Command(long mResult);
Boolean Do_Command(long mResult);


extern void start_running(void);
extern void stop_running(void);
extern void do_show_lut(unsigned long base);
extern void do_nice_exit(void);
extern void do_image_adjust(void);
extern void do_roi_adjust(void);
extern void do_valve_adjust(void);
extern void do_image_mapping(void);
extern void do_image_adjust(short vd_number, short *br, short *cntr, short *hue, short *sat);
extern void do_valve_control(Point where);
extern void save_settings(void);
extern void do_shadow_control(unsigned char mode);
extern void do_pattern_control(unsigned char mode);
extern void do_run_mode_control(unsigned char mode);
extern void do_simulated_speed_control(void);

// -----------------------------------------------------------------------------

void EventTask()
{
     unsigned char  din_c;
     static unsigned char last_din_c;

     do
     {
///*
        // check switch settings
        read_dio(C, &din_c);

        if((din_c & 0xf0) != last_din_c)
        {
          // update digital input record
             last_din_c = din_c & 0xf0;

          // automatic / manual mode switch
             do_run_mode_control(last_din_c & AUTO_MODE);

          // extend pattern switch
             do_pattern_control(last_din_c & EXPAND_PATTERN);

          // spray shadows switch
             do_shadow_control(last_din_c & SPRAY_SHADOWS);
        }
        //read_dio(C, &din_c);
        //status.expand = din_c && EXPAND_PATTERN;

        // manual / auto switch
          ///if(din_c & AUTO_MODE)
             ///status.run = YES;
///myprint(status_line, "0E", 0);
//*/
        WaitNextEvent(everyEvent, &gEvent, LONG_NAP, NO_CURSOR);
        if(gEvent.what)
          DoEvent();
     } while(!status.run);
}

// -----------------------------------------------------------------------------

// check for user input
```

```
void DoEvent()   //call appropriate routines
{
   char    key;

   switch ( gEvent.what )
   {
     // mouse action
     case mouseDown:
       gwindowCode = FindWindow(gEvent.where, &gwhichWindow);
         switch(gwindowCode)
         {
           case inMenuBar:
             guserDone = Do_Command(MenuSelect(gEvent.where));
             break;
           //case inSysWindow:
             //SystemClick(&gEvent, gwhichWindow);
             //break;
           case inContent:
             buttons(gEvent.where);
             break;

         }
     // keyboard input
      case keyDown:
        case autoKey:
          if((gEvent.modifiers & cmdKey) != 0)
            guserDone = Do_Command(MenuKey((char)(gEvent.message & charCodeMask)));
          else
        {
        key = gEvent.message & charCodeMask;
        switch (key)

        {
          case ' ':
            do_run_mode_control(!status.run);
            break;
          }
        }
   }

} //End of DoEvent
// ---------------------------------------------------------------------------------

void buttons(Point where)
{
   short i;

   // console controls
     if(mouse_in(mode_rect, where))
       do_run_mode_control(!status.run);

     if(mouse_in(shadow_rect, where))
       do_shadow_control(status.shadows);

     if(mouse_in(pattern_rect, where))
       do_pattern_control(status.expand);

   // computer controls
     /////for(i = 0; i<10; i++)
     /////{
     /////  if(mouse_in(simSpeedRect[i], where))
     /////  {
     /////    simSpeed = (unsigned long)(i + 1);
     /////    do_simulated_speed_control();
     /////  }
     /////}

     /////if(mouse_in(radar_rect, where))
     /////  if(status.distance_counter)         // switch to simulated travel
     /////  {
     /////    my_draw_frame_in(&radar_rect, 3);
     /////    my_draw_rect(radar_rect, BLACK);
     /////    status.distance_counter = OFF;
     /////    firstTick = TickCount();
     /////  }
     /////  else                               // switch to radar measurement
     /////  {
     /////    draw_depressed_button(radar_rect);
     /////    status.distance_counter = ON;
     /////    overflow_record += ((float)(TickCount() - firstTick) / 65536.0);
     /////    read_counter(B2, &last_count);
     /////  }

     if(mouse_in(do_show_data_rect, where))
       if(status.show_data == YES)
       {
         my_draw_frame_in(&do_show_data_rect, 3);
         my_draw_rect(do_show_data_rect, BLACK);
         status.show_data = NO;
       }
       else
       {
         draw_depressed_button(do_show_data_rect);
         status.show_data = YES;
       }

     if(mouse_in(do_show_pixels_rect, where))
       if(status.show_pixels == YES)
       {
         my_draw_frame_in(&do_show_pixels_rect, 3);
         my_draw_rect(do_show_pixels_rect, BLACK);
         status.show_pixels = NO;
       }
       else
       {
         draw_depressed_button(do_show_pixels_rect);
         status.show_pixels = YES;
       }

   // configuration editing controls
     if(mouse_in(do_vd_edit_rect, where))
       do_image_adjust(0, &vd[0].brightness, &vd[0].contrast, &vd[0].hue, &vd[0].saturation);

     if(mouse_in(do_roi_edit_rect, where))
       do_roi_adjust();

     if(mouse_in(do_valve_edit_rect, where))
       do_valve_adjust();

     if(mouse_in(valve_control_rect, where))
       do_valve_control(where);

     if(mouse_in(do_image_map_rect, where))
       do_image_mapping();

     if(mouse_in(do_save_rect, where))
       save_settings();
}
// ---------------------------------------------------------------------------------
short mouse_in(Rect c_rect, Point click)
{
  if(click.h > c_rect.left && click.h < c_rect.right && click.v > c_rect.top &&
     click.v < c_rect.bottom)
     return(1);
  else
     return(0);
}
// ------------------------------------------------------------------------
Boolean Do_Command(long mResult)
{
    unsigned char  accName[255];
    short          itemHit;
    Boolean        quitApp;
    short          mrefNum;
    DialogPtr   theDialog;
    short          theItem, theMenu;
    GrafPtr     savePort;

    quitApp = FALSE;
    theMenu = HiWord(mResult);
    theItem = LoWord(mResult);

    switch (theMenu)
    {
      case APPLE_MENU:
        if(theItem == ABOUT_BOX)
        {
          if((theDialog = GetNewDialog(ABOUT_BOX_ID, NIL, (WindowPtr) IN_FRONT)) != NIL)
          {
            ModalDialog(NIL,&itemHit);
            DisposeDialog(theDialog);
          }
          else
            SysBeep(1);
        }
        else
        {
          GetPort(&savePort);
          GetMenuItemText(gmyMenus[(APPLE_MENU - RESOURCE_ID)], theItem, accName);
          mrefNum = OpenDeskAcc(accName);
          SetPort(savePort);
        }
      case CONTROL_MENU:
        switch(theItem)
        {
          case RUN_NOW:
            start_running();
            break;
          case STOP_NOW:
            stop_running();
            break;
          case QUIT_NOW:
            do_nice_exit();
            break;
        }
      break;
      case IMAGE_MENU:
        switch(theItem)
        {
          case SHOW_LUT:
            do_show_lut(vd[0].base);
            break;
          case LOAD_LUT:
            // not implemented yet
            SysBeep(1);
            break;
          case ADJUST_IMAGE:
            do_image_adjust(0, &vd[0].brightness, &vd[0].contrast, &vd[0].hue, &vd[0].saturation);
            break;
          case MAP_IMAGE:
            do_image_mapping();
            break;
          case COPY_IMAGE:
            grab_to_clip(0, 0, 640, 480);
            break;
        }
      break;
      case VALVE_MENU:
        break;
      case RUN_MODE_MENU:
        switch(theItem)
        {
          case PIXEL_DISPLAY:
            status.show_pixels = (status.show_pixels) ? 0 : 1;
          break;
          case DATA_DISPLAY:
            status.show_data = (status.show_data) ? 0 : 1;
            if(status.show_data == 1)
              draw_data_area(ON);
            else
              draw_data_area(OFF);
          break;
        }
    }
    HiliteMenu(0);
    return quitApp;
  }
// ------------------------------------------------------------------------
// ------------------------------------------------------------------------
```