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

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>LIST OF FIGURES</td>
<td>1</td>
</tr>
<tr>
<td>AHMCT GOALS &amp; DISCLOSURE STATEMENT</td>
<td>2</td>
</tr>
<tr>
<td>INTRODUCTION</td>
<td>3</td>
</tr>
<tr>
<td>OBJECTIVE</td>
<td>4</td>
</tr>
<tr>
<td>METHODS &amp; MATERIALS</td>
<td>5</td>
</tr>
<tr>
<td>- Computer Vision System</td>
<td>5</td>
</tr>
<tr>
<td>- Rapid Response Spray System</td>
<td>7</td>
</tr>
<tr>
<td>- Intelligent Intermittent Spray System Operation</td>
<td>8</td>
</tr>
<tr>
<td>- User Interface</td>
<td>9</td>
</tr>
<tr>
<td>- IISS Operating Instructions</td>
<td>10</td>
</tr>
<tr>
<td>- List of IISS components</td>
<td>11</td>
</tr>
<tr>
<td>- Wiring Diagrams</td>
<td>12</td>
</tr>
<tr>
<td>TESTING &amp; PERFORMANCE</td>
<td>17</td>
</tr>
<tr>
<td>CONCLUSIONS &amp; RECOMMENDATIONS</td>
<td>17</td>
</tr>
<tr>
<td>REFERENCES</td>
<td>18</td>
</tr>
<tr>
<td>APPENDIX - IISS COMPUTER CODE</td>
<td>19</td>
</tr>
<tr>
<td>Figure</td>
<td>Description</td>
</tr>
<tr>
<td>----------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Figure 1</td>
<td>Photograph of the Intelligent Intermittent Spray System.</td>
</tr>
<tr>
<td>Figure 2</td>
<td>Schematic diagram showing the camera field of view and individual regions of interest.</td>
</tr>
<tr>
<td>Figure 3</td>
<td>Schematic diagram of the rapid-response intermittent spray system.</td>
</tr>
<tr>
<td>Figure 4</td>
<td>Schematic diagram of the prototype Intelligent Intermittent Spray System.</td>
</tr>
<tr>
<td>Figure 5</td>
<td>Flowchart showing the sequence of events for operating the IISS.</td>
</tr>
<tr>
<td>Figure 6</td>
<td>Layout of control panel and operating instructions</td>
</tr>
<tr>
<td>Figure 7</td>
<td>Wiring diagram for in-cab console unit</td>
</tr>
<tr>
<td>Figure 8</td>
<td>Wiring diagram for interface board (1)</td>
</tr>
<tr>
<td>Figure 9</td>
<td>Wiring diagram for interface board (2)</td>
</tr>
<tr>
<td>Figure 10</td>
<td>Schematic for power and solenoids</td>
</tr>
<tr>
<td>Figure 11</td>
<td>Schematic for interconnect cables</td>
</tr>
</tbody>
</table>
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. 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.

![Photograph of the intelligent intermittent spray system.](image)

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 maintenance 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.

![Diagram of Intelligent Intermittent Spray System](image)

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.
IISS Operating Instructions

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.

![Control Panel Diagram]

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


## APPENDIX - IISS COMPUTER CODE

<table>
<thead>
<tr>
<th>File Name</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>TAOS-V1.0</td>
<td>Main program</td>
<td>1</td>
</tr>
<tr>
<td>TAOS.H</td>
<td>Control panel</td>
<td>15</td>
</tr>
<tr>
<td>TAOS_RSRC.H</td>
<td>Menu options</td>
<td>15</td>
</tr>
<tr>
<td>ROPS_CONTROL.H</td>
<td>Controls video digitizer</td>
<td>16</td>
</tr>
<tr>
<td>TAOS.Utility.H</td>
<td>Utility functions for MVS control program</td>
<td>19</td>
</tr>
<tr>
<td>TAOS_IM_ANALY.H</td>
<td>Analyzes video images</td>
<td>24</td>
</tr>
<tr>
<td>MAC_INI.H</td>
<td>Initializes the Macintosh</td>
<td>30</td>
</tr>
<tr>
<td>TAOS.VALVES.H</td>
<td>Controls the valves</td>
<td>31</td>
</tr>
<tr>
<td>TAOS_DISPLAY.H</td>
<td>Displays the TAOS control information</td>
<td>33</td>
</tr>
<tr>
<td>TAOS_INTERFACE.H</td>
<td>Controls the TAOS system using computer input</td>
<td>44</td>
</tr>
</tbody>
</table>
// System functions
short vd_init[0](video_digitzer *vd);
short get_roi_data[0](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[0](roi zone[MAX_SET+1][MAX_BAND+1], short *band_count, short *set_count,
   unsigned long base);
void get_distance[0](unsigned long *distance);
void do_image_adjust[0](short vd_number, short *brightness, short *contrast, short *hue,
   short *sat);
void do_roi_adjust[0](void);
void view_start_running[0](vd id);
void stop_running[0](void);
void draw[0](unsigned long base);
void do[of the][0](void);
void show_image[0](unsigned long plant[]), int first_set, int last_set);
void do_valves[0](valves valve[]), unsigned long plant[]), int speed, short call_index,
   int valve_action_increment);
void send_valve_signal[0](unsigned long valve_signal);
void show_valve_signal[0](unsigned long valve_signal);
void show_speed[0](float speed);
void do_image_mapping[0](void);
void get_valve_data[0](valve[]), short *valve_count);
void update_roi_display[0][0](Rect *rect, Rect *rect, char *string);
void save_settings[0](void);
void do_valve_adjust[0](void);
short set_video_display[0](video_digitzer *vd, short width, short height, Point start,
   short mode); 
void do_run_mode_control[0](unsigned char mode);
void do_pattern_control[0][0](unsigned char mode);
void do_shadow_control[0][0](unsigned char mode);
extern void EventTask[0](void);

// ----- start of main function ----- 
int main(void)
{
    // variables
    unsigned char *feature = 0;
    char string[100] = 0;
    char infile[20] = 0;
    int set = -1, band = -1;
    int err = 0, location = 1;
    int volumn = 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_snow=0.0;
    FILE *in_file;
    int loop = 0;
// set mainstash operating parameters
initialize_mac();

// find v4l sock / address
if(!openVideo(&vd[0].refsum, &vd[0].base))
    SysBeep1();

// set up display window and misc.
SetRect(&main_win_w, main_win_h, WINDOW_LEFT, WINDOW_TOP, 830, 620);
mainWindow = NewCWWindow(NULL, &main_win_w, "pTarget Activated Offset Sprayer (TASO)
true, &pGrowBoxProc, (WindowProc(-1)), false, NULL);
initialize_display(1,1);
draw_main_screen();

// get control width
myprint(status_line, "Reading configuration file", 0);
string(in_file, "main.ini");
if(!file_open(in_file, "r"))
    {
        while(getcstring(string, "CONTROL WIDTH", 13))
            gets(string, 100, in_file);
        fscanf(in_file, "%f", &control_width);
        fclose(in_file);
    }
else
    {
        sprintf(string, "File missing: ".in", in_file);
        myprint(status_line, string, 0);
        pass(5.0);
        exit(0);
    }

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

// get image width
if(!file_open("main.js", "r"))
    {
        while(getcstring(string, "IMAGE WIDTH", 11))
            gets(string, 100, in_file);
        fscanf(in_file, "%d", &max_image_w);
        fclose(in_file);
    }

// get image coefficients
if(!file_open("main.js", "r"))
    {
        while(getcstring(string, "VDI IMAGE COEFFICIENTS", 22))
            gets(string, 100, in_file);
        fscanf(in_file, "%f", &vd[0].h_coef);
        fscanf(in_file, "%f", &vd[0].v_coef);
        fclose(in_file);
    }

// get vd settings
if(!file_open("main.js", "r"))
    {
        while(getcstring(string, "VDI SETTINGS", 13))
            gets(string, 100, in_file)
        fscanf(in_file, "%f", &brightness);
        fscanf(in_file, "%f", &contrast);
        fscanf(in_file, "%f", &hue);
        fscanf(in_file, "%f", &saturation);
        fscanf(in_file, "%f", &width);
        fscanf(in_file, "%f", &height);
        fscanf(in_file, "%f", &params);
        fclose(in_file);
    }

vd[0].brightness = (short)brightness;
v0.contrast = (short)contrast;
v0.hue = (short)hue;
v0.saturation = (short)saturation;
myprint(status_line, "Configuration finished", 0);

// initialize screen
if(!dc_init(0, &main_win_w, &main_win_h))
    {
        printf("Initialize screen failed\n");
        exit(0);
    }

// myprint(status_line, "LOO Initialized", 0);

// initialize vdeo digitizer
if(err = vd_init(&vd[0],
    myprint(status_line, "Video digitizer initialization error: ", err); exit(0);

    myprint(status_line, "Video digitizer initialized", 0);

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

// define maximum image offset allowed (ravens)
max_offset = vd[0].disp_rect[2] - vd[0].disp_rect[3] - (set_count) * (zone[0][0] - zone[0][0]) + (zone[0][0] - zone[0][0]);
max_offset = 0;

// read video information
get_video_data(valve, &valve_count);

// get color look-up table
myprint(status_line, "Loading CLUTs", 0);
if(!get_color_table())
    {
        SysBeep(0);
        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 = (vd[0].image_rect_right - (vd[0].image_rect_left + 80));

if(!draw_set_rect(&vd[0].params, &vd[0].width, &vd[0].height, 11))
    {
        myprint(status_line, "vd error: ", err);
        exit(0);
    }

// 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.no = NO;
status.distance_counter = ON;
myprint(status.Line, "Initializng 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, &in_c);
    write Dio(c, in_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]); // frame grab with vd
        if(low_framrate) // first slot with vd
            get_distance(&fg_start_time);
        else
            get_distance(&current_dist);
        next_fg_dist = fg_start_time + set_count * valve_action_increment;

        // re-initialize for new loop
        next_valve_action_dist = fg_start_time + 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 = ((TickCount() - last_fg_start_time) / (TickCount() - last_fg_start_time)); // (raves / 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;
            speed_sum += speed_now; // add new instantaneous speed
            speed = speed_sum / 5.0; // divide to get raves per tick
        }
    }

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

// wait for frame to finish
while(check_frame_grab(vd[0], &active))
{
    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(low_framrate) // (next_valve_action_dist + valve_allowance)
        do_overspeed_warning(0); // too far - gap in spray coverage
    else
        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);

data

// analyze image

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

// 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;
    do_overspeed_warning(3); // too many frame-grab delays - inaccurate
}

// do each set
for(set = 0; set < set_count; set++)

    // go do roi analysis
    if(set & roi)
        do_valves(valve_plants, speed, call_index, valve_action_increment);
    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);
    do_overspeed_warning(0); // too far - gap in spray coverage
    call_index++; // number of valve controls so far this loop
myprintf(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(VALUE5_OFF);
|
| // do mapping
| draw_image_map_screen(ON);
| do_image_map(); // see taos_radar.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_consols(ON);
|
| // -----------------------------------------------
| // ----- display the current CLUT and allow it to be saved -----

void do_show_list(unsigned long base)
|
| // stop control loop
| status.run = OFF;
| send_valve_signal(VALUE5_OFF);
|
| // display the CLUT
| show_list(base); // see taos_radar.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_consols(ON);
|
| // -----------------------------------------------
| // ---- tidy up before quitting (not used from operator console) ----

void do_tidy_up(void)
|
| short vd_number = 0;
| WindowPr current_window;
|
| // tidy up before quitting
| send_valve_signal(VALUE5_OFF);
| myprintf(status_line, "resetting video digitizer", 0);
| vd.reset(vd_number parm); while(current_window = FromWindow0)
| 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(82, count);   // see lab_ab_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 tick
  write_dio(C, din_c | WATCHDOG);
}

void do_valve_control(void mouse_loc)
{

}

/*
  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 talk_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);
  }
*/
valve[v].set_status(MANUAL_ON);
send_valve_signal(0xffff);
display_valve_control(v, MANUAL_ON);
show_valve_signal(0xffff);

break;

// turn off the valve
if(mouse_in(all_off_rect, mouseLoc))
{
  for(v=0; v < MAX_VALUE; 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 width, height, skip = 1;
  left = data_rect.left + (width+1) + 1;
  for(band = 0; band < band_count; band++)
  {
    left += width + 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, 0xff000000, skip);
      }
      else
      {
        my_fill_rect(&plant_rect, 0x00555566, skip);
      }
    }
  }
}

// display valve control signal
void show_valve_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++)
  {
    my_fill_rect(&signal_rect, color, skip);
  }
}

// display travel speed
void show_speed(float speed) // speed is in raven's/dick
{
  char string[10] = "0";
  static Rect bar_rect;

  if(bar_rect.top + speed * 10) // do bar indicator
  {
    bar_rect.top = speed * 10;
    bar_rect.right = bar_rect.left + 5;
    my_fill_rect(&bar_rect, 0x0000ff00, 1);
  }

  if(bar_rect.top + speed * 10) // display speed
  {
    sprintf(string, "%.3f miles\n" + speed * 0.462); // display mph
    my_fill_rect(&speed_tiles, WHITE, 1);
  }

  // display speed
  sprintf(string, "%.3f mph", speed*1.032); // display mph
  my_fill_rect(&speed_tiles, WHITE, 1);
}
// ...initialize video digitizer (see ROps_control.h) ...

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

    myprint(status_line, "Opening video digitizer driver", 0);
    if(err = open_video(&(*vd).refnum, &(*vd).base))
        return(err);
    myprint(status_line, "Selecting video digitizer", 0);
    if(err = 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 = 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, &max_width, &max_height))
        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 - 8;
    (*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 - 8;


    (*vd).startx = VD_1_WINDOW_TOP;
    (*vd).starty = VD_1_WINDOW_LEFT;

    myprint(status_line, "Setting video digitizer size", 0);
        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_subbus(&(*vd).params, 255))
        return(err);

    return(err);
}

// ...set up display window ...

short set_video_display(video_digitizer *vd, short width, short height, short point, 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, x, y))
        return(err);

    return(0);
}

// ...read valve information from configuration file ...

void get_valve_data(char *valvevalvevalvevalve, short *valve_count)
{
    char in[200] = {0};
    FILE *in_file;
    short num;

    myprint(status_line, "opening valve file", 0);
    if(is_open = fopen("in.txt", "r"))
    {
        while(strcmp(in, "VALVE", 3))    // read through file line by line until
            getstring(200, in_file);    // valve information title found
        getstring(200, in_file);    // read valve data heading
        num = 0;
    while(1)
        if(strcmp(in, "END", 3))    // read valve data until END
            getstring(200, in_file);
        else
            else myprint(status_line, "Valve file not read", 0);
        if(*valve_count == MAX_VALVE)
            myprint(status_line, "Valve count exceeded, check configuration file", 0);
            num = 0;
            pause(0);
            exit(0);
        }

    // ...read the roi data is from configuration file ...

    short get_roi_data(char zone[1],.short *valve_count, int control_increment,
        float band) short *band_count, unsigned long base)
    {
        short err = 0;

        Copyright 2011, AHMCT Research Center, UC Davis
int bad = 0;
float set_width;
FILE *io_file;
char string[200] = {0};
float rol_offset;
short vd_number = 0;

// find width of set in pixels (nearest band)
set_width = (float)control_increments / RAEN

// find meters to middle of image
rol_offset = (float)(vd_number) * disp_rect.bottom

// read data from file
if (in_file = fopen("xiao", "r"))

// read leading roi data for each band in file
while(1)

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

// define remaining regions
make_rois(zone, band_count, set_count, base);
return(0);

// define additional roi's in band on first set

short make_rois(zone[0][1], MAX_BAND+1), short *band_count, short *set_count,
unsigned long base

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

// find locations for even / odd display windows
height = vd_number; height = 1;
start_y = vd_number.start_y = height;

// draw video digitizer adjustment window and controls
draw_physical(vd_number, refnum, WAIT);
status.run = OFF;

my_draw_x_v((int)v_d, *contrast_rect, *contrast, 0, 255);
my_draw_y_v((int)v_d, *brightness, 0, 255);
my_draw_x_level(vd_hue_rect, *hue, 0, 255);
my_draw_x_level(vd_sat_rect, *saturation, 0, 255);
if(vd_number.source == S_VIDEO)
    my_fill_rect(&as_video_rect, BLACK, 1);
    my_fill_rect(&composite_rect, WHITE, 1);
else
    my_fill_rect(&as_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, &Event, LONG_NAP, NO_CURSOR);
    if(Event.what == mouseDown)
        GetMouse(AMouseloc);
        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); //vd_contrast_rect.bottom - vd_contrast_rect.top;
        else if(mouse_in(vd_more_brightness_rect, mouseloc))
            *brightness = 1;
        else if(mouse_in(vd_less_brightness_rect, mouseloc))
            *brightness = -1;
        else if(mouse_in(vd_brightness_rect, mouseloc))
            *brightness = 255 * (vd_brightness_rect.bottom - mouseloc); //vd_brightness_rect.bottom - vd_brightness_rect.top;
        else if(mouse_in(vd_hue_rect, mouseloc))
            *hue = 255 * (vd_hue_rect.bottom - mouseloc); //vd_hue_rect.bottom - vd_hue_rect.top;
        else if(mouse_in(vd_sat_rect, mouseloc))
            *saturation = 255 * (vd_sat_rect.bottom - mouseloc); //vd_sat_rect.bottom - vd_sat_rect.top;
        else if(mouse_in(s_video_rect, mouseloc)) // set for s-video input
            HideCursor();
            vd_number.source = S_VIDEO;
            my_fill_rect(&as_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_number.source = COMPOSITE;
            my_fill_rect(&as_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(&applylut_rect, mouseloc)) // apply the CLUT to image
            HideCursor();
            lut_flag = (lut_flag == 0 ? 1 : 0);
            if(lut_flag)
                my_fill_rect(&applylut_rect, BLACK, 1);
            else
                my_fill_rect(&applylut_rect, WHITE, 1);
                cont_frame(vd[0].params, DONTWAIT);
                ShowCursor();
        // update values
        vd_number.brightness = *brightness;
        vd_number.contrast = *contrast;
        vd_number.hue = *hue;
        vd_number.saturation = *saturation;
        // separate odd / even display
        if(split_flag)
            set_video_display(&vd[0].width, vd[0].height, vd[0].start, 1);
            grab_frame(1, vd[0].refnum, WAIT);
            set_video_display(&vd[0].width, vd[0].height, start, 2);
            grab_frame(1, vd[0].refnum, WAIT);

    // do adjustments to digitizer board
    if(evm[vd_set_video_level].params & vd_number.brightness,
       &vd_number.contrast)
        myprint(status_line, "video digitizer error ", err);
    if(evm[vd_set_hue_level].params & vd_number.hue)
        myprint(status_line, "video digitizer error ", err);
    if(evm[vd_set_sat_level].params & vd_number.saturation)
        myprint(status_line, "video digitizer error ", err);
    if(evm[vd_set_source].params & vd_number.source)
        myprint(status_line, "video digitizer error ", err);
    // update control bar levels in display
    my_draw_x_level(vd_contrast_rect, *contrast, 0, 255);
    my_draw_x_level(vd_brightness_rect, *brightness, 0, 255);
    my_draw_x_level(vd_hue_rect, *hue, 0, 255);
    my_draw_x_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 test results
if (test_flag)

    grab_frame(1, vd[0].refnum, WAIT);
    big_zone_analyze_roi(0, 1);

// done - reset things and clean up
vd[0].start = VD_1_WINDOW_TOP;
vd[0].start = VD_1_WINDOW_LEFT;
vd[0].image_rect.right = zone[0][band_count].center + 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_vo_adjust_area(OFF);

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

grab_frame(1, vd[0].refnum, WAIT);

// pattern control
void do_pattern_control(unsinged 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(unsinged char mode)
{
    HideCursor();
    if (mode)
        stop_running();
        my_fill_rect(&mode_control_rect, WHITE, 1);
    else
        scan_running();
        my_fill_rect(&mode_control_rect, BLACK, 1);
    ShowCursor();
}

// --------------------------------------------
// ---- run mode control
// --------------------------------------------

// --------------------------------------------
// ---- modify the roi definitions
// --------------------------------------------

void do_vo_adjust()
{
    short en = 0;
    Point mouseloc;
    int i, j, value;
    Rect rect;
    float fvalue;
    char string[50] = "{0};
    short dir;
    short set, shift;
    short band;
    float rel_voffset;
    short vd_number = 0;

    // set up roi adjustment window
    status.run = OFF;
    draw_main_screen();
    draw_vo_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].center + pixel();
        if (set = 0, set < set_count, set++)
            zone[set][band].reset_roi(shift, vd[0].base);

    // find mouse click and proceed with edit
    while ((mouse_in roi, finished rect, mouseloc))
    {
        WaitNextEvent(everyEvent, &eEvent, LONG_NAP, NO_CURSOR);
        if (eEvent.what == mouseDown)
            GetMouse(&mouseloc);
            for (i = 0; i < band_count, i++)
                do_vo_adjust();
            }
if (mouse_in(&hand_up[i], mouseloc))
	value = 0;
b else if (mouse_in(&hand_down[i], mouseloc))
	value = -1;
break;
if (value < 0)
	value = 0;
if (value > 0)
	value = 0;
while (StillDown(0))
{
	if (dir < band_count)
	{
		if (mouse_in(&near_side_rect, mouseloc))
		{
			value = zone[0][i].roi_nearside();
			do
			{
				value += 0.01 * dir;
				if (value < 0.0)
					value = 0.0;
				else
					value = 0.0;
				sp艇lprintf("%3.2f", value);
				update_roi_display(&near_side_rect, &band_rect[i], string);
				zone[0][i].edit_roi_nearside(value, vd[0].base, &band_of(set_adj[i]));
			pause(0.1);
			while (StillDown(0));
		}
		if (mouse_in(&far_side_rect, mouseloc))
		{
			value = zone[0][i].roi_farside();
			do
			{
				value += 0.01 * dir;
				if (value < 0.0)
					value = 0.0;
				else
					value = 0.0;
				sp艇lprintf("%3.2f", value);
				update_roi_display(&far_side_rect, &band_rect[i], string);
				zone[0][i].edit_roi_farside(value);
			pause(0.1);
			while (StillDown(0));
		}
		if (mouse_in(&horz_rect, mouseloc))
		{
			value = zone[0][i].roi_horz_skip();
			do
			{
				value += 1 * dir;
				if (value < 1)
					value = 1;
				else
					value = 1;
				sp艇lprintf("%d", value);
				update_roi_display(&horz_rect, &band_rect[i], string);
				zone[0][i].edit_roi_horz_skip(value);
			pause(0.1);
			while (StillDown(0));
		}
		if (mouse_in(&vert_rect, mouseloc))
		{
			value = zone[0][i].roi_vertSkip();
			do
			{
				value += 1 * dir;
				break;
			}
			if (mouse_in(&noise_rect, mouseloc))
		{
				value = zone[0][i].roi_noise();
			do
			{
				value += 1 * dir;
				if (value < 0)
					value = 0;
				else
					value = 0;
				sp艇lprintf("%d", value);
				update_roi_display(&noise_rect, &band_rect[i], string);
				zone[0][i].edit_roi_noise(value);
			pause(0.1);
			while (StillDown(0));
		}
		if (mouse_in(&more_control_width_rect, mouseloc))
		{
			do
			{
				if (mouse_in(&more_control_width_rect, mouseloc))
					pause(action_increment + 1);
				else
					pause(action_increment - 1);
				sp艇lprintf("Control Width: \%3.2f", (float)pause(action_increment);
				my_fill_rect(&control_width_rect, WHITE, 1);
				my_draw_rect(&control_width_rect, BLACK);
				pause(0.1);
			while (StillDown(0));
			for (x0 < i < band_count, i++)
			none[0][i].edit_control_width(pause(action_increment));
		}
		if (mouse_in(&more_image_rect, mouseloc))
		{
			do
			{
				my_fill_rect(&less_image_rect, 0x00000000, 2);
				my_fill_rect(&more_image_rect, 0x00000000, 2);
				if (mouse_in(&less_image_rect, mouseloc))
				
taxmax:image_y = 1;
				else if (mouse_in(&more_image_rect, mouseloc))
					max_image_y = 1;
				sp艇lprintf("Image Width: \%d pixels", max_image_y);
				my_fill_rect(&image_width_rect, WHITE, 1);
				my_draw_rect(&image_width_rect, BLACK);
				pause(0.1);
			while (StillDown(0));
			max_offset = (v0/window_top + v0.image_rect.bottom - 4 * max_image_y)
			}"}
if (mouse_in(add_roi_rect, mouseloc))
{
    band_count++;
    sprintf(string, "%d %d %d %d", band_count-1, zone[0][band_count-2].roi_top, zone[0][band_count-2].roi_bot, zone[0][band_count-2].roi_left);
    zone[0][band_count-2].roi_top = zone[0][band_count-2].roi_bot;
    zone[0][band_count-2].roi_bot = zone[0][band_count-2].roi_top + 20;
    zone[0][band_count-2].roi_left = zone[0][band_count-2].roi_right;
    zone[0][band_count-2].roi_right = zone[0][band_count-2].roi_left + 20;
    zone[0][band_count-2].roi_top = zone[0][band_count-2].roi_bot - 20;
    zone[0][band_count-2].roi_bot = zone[0][band_count-2].roi_top + 20;
    zone[0][band_count-2].roi_left = zone[0][band_count-2].roi_right;
    zone[0][band_count-2].roi_right = zone[0][band_count-2].roi_left + 20;
    ro1_voffset = (float)(vd[0].disp_rect_d.bottom - vd[0].disp_rect_d.top) / (vd[0].disp_rect_d.right - vd[0].disp_rect_d.left);
    zone[0][band_count-1].set_roi(string, (float)valve_action_increment / RAVERS,
&band_offset_adj[band_count-1], vd[0].base, ro1_voffset);
    shift = zone[0][band_count-1].find_min_vpixel();
    zone[0][band_count-1].rect_roi(shift, vd[0].base);
    rect.top = band_rect[band_count-2].top;
    rect.left = band_rect[band_count-2].left;
    rect.right = band_rect[band_count-2].right + 5;
    rect.bottom = delete_roi_rect.bottom + 5;
    initialize_display(band_count, valve_count);
}

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(Rec, MAIN_BG, 1);
    draw_roi_adjust_area ON, band_count, zone; ShowCursor();
}

if (mouse_in(add_roi_rect, mouseloc))
{
    band_count++;
    sprintf(string, "%d %d %d %d", band_count-1, zone[0][band_count-2].roi_top, zone[0][band_count-2].roi_bot, zone[0][band_count-2].roi_left);
    zone[0][band_count-2].roi_top = zone[0][band_count-2].roi_bot;
    zone[0][band_count-2].roi_bot = zone[0][band_count-2].roi_top + 20;
    zone[0][band_count-2].roi_left = zone[0][band_count-2].roi_right;
    zone[0][band_count-2].roi_right = zone[0][band_count-2].roi_left + 20;
    zone[0][band_count-2].roi_top = zone[0][band_count-2].roi_bot - 20;
    zone[0][band_count-2].roi_bot = zone[0][band_count-2].roi_top + 20;
    zone[0][band_count-2].roi_left = zone[0][band_count-2].roi_right;
    zone[0][band_count-2].roi_right = zone[0][band_count-2].roi_left + 20;
    ro1_voffset = (float)(vd[0].disp_rect_d.bottom - vd[0].disp_rect_d.top) / (vd[0].disp_rect_d.right - vd[0].disp_rect_d.left);
    zone[0][band_count-1].set_roi(string, (float)valve_action_increment / RAVERS,
&band_offset_adj[band_count-1], vd[0].base, ro1_voffset);
    shift = zone[0][band_count-1].find_min_vpixel();
    zone[0][band_count-1].rect_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);
}

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, (*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);
}

void do_value_adjust()
{
    short i, dir, int valore;
    float value;
    Point mouseloc;
    char string[30] = "0"

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

    // modify settings
    while (mouse_in(edi_edit_completed, mouseloc))
    {
        WinNextEvent(everyEvent, &eEvent, LONG_NAP, NO_CURSOR);
        if (eEvent.what == mouseDown)
        {
            GetMouse(&mouseloc);
            for (i = 0; i < valve_count; i++)
            {
                if (mouse.in(edi_up[i][j], mouseloc))
                {
                    dir = 1;
                    break;
                }
                else if (mouse.in(edi_down[i][j], mouseloc))
                {
                    dir = -1;
                    break;
                }
            }
        }
    }
}
if(i < valve_count)
{
    // adjust distance from camera to valve
    if(mouse_in(dist_rect, mouseloc))
    {
        value = val[i].get_dist();
        do
        {
            value += 0.01 * dir;
            if(value < 0.0)
                value = 0.0;
            sprintf("%.3f", value);
            update_roi_display(&dist_rect, &edit_rect[i], string);
            val[i].edit_dist(value);
            pause(0.1);
        } while(StillDown());
    }

    // adjust time of flight (tof) adjustment
    if(mouse_in(tof_rect, mouseloc))
    {
        value = val[i].get_tof();
        do
        {
            value += 0.005 * dir;
            if(value < 0.0)
                value = 0.0;
            sprintf("%.3f", value);
            update_roi_display(&tof_rect, &edit_rect[i], string);
            val[i].edit_tof(value);
            pause(0.1);
        } while(StillDown());
    }

    // adjust f_lead
    if(mouse_in(f_lead_rect, mouseloc))
    {
        value = val[i].get_f_lead();
        do
        {
            value += 0.005 * dir;
            if(value < 0.0)
                value = 0.0;
            sprintf("%.3f", value);
            update_roi_display(&f_lead_rect, &edit_rect[i], string);
            val[i].edit_f_lead(value);
            pause(0.1);
        } while(StillDown());
    }

    // adjust f_lag
    if(mouse_in(f_lag_rect, mouseloc))
    {
        value = val[i].get_f_lag();
        do
        {
            value += 0.005 * dir;
            if(value < 0.0)
                value = 0.0;
            sprintf("%.3f", value);
            update_roi_display(&f_lag_rect, &edit_rect[i], string);
            val[i].edit_f_lag(value);
            pause(0.1);
        } while(StillDown());
    }
}

// adjust d_lead
if(mouse_in(d_lead_rect, mouseloc))
{
    value = val[i].get_d_lead();
    do
    {
        value += 0.005 * dir;
        if(value < 0.0)
            value = 0.0;
        sprintf("%.3f", value);
        update_roi_display(&d_lead_rect, &edit_rect[i], string);
        val[i].edit_d_lead(value);
        pause(0.1);
    } while(StillDown());
}

// adjust f_lag
if(mouse_in(f_lag_rect, mouseloc))
{
    value = val[i].get_f_lag();
    do
    {
        value += 0.005 * dir;
        if(value < 0.0)
            value = 0.0;
        sprintf("%.3f", value);
        update_roi_display(&f_lag_rect, &edit_rect[i], string);
        val[i].edit_f_lag(value);
        pause(0.1);
    } while(StillDown());
}

// adjust band (inside)
if(mouse_in(band_in_rect, mouseloc))
{
    value = val[i].get_band_in();
    do
    {
        value += 1 * dir;
        if(value < 0.0)
            value = 0.0;
        sprintf("%.3f", value);
        update_roi_display(&band_in_rect, &edit_rect[i], string);
        val[i].edit_band_in(value);
        pause(0.1);
    } while(StillDown());
}

// adjust band (outside)
if(mouse_in(band_out_rect, mouseloc))
{
    value = val[i].get_band_out();
    do
    {
        value += 1 * dir;
        if((value < 0.0)
            value = 0.0;
        sprintf("%.3f", value);
        update_roi_display(&band_out_rect, &edit_rect[i], string);
        val[i].edit_band_out(value);
        pause(0.1);
    } while(StillDown());
}
// doo - reset the main display window
draw_main_screen();
draw_video_area(ON);
draw_data_area(ON);
draw_overhead_area(ON);
draw_speep_area(ON);
draw_controls(ON);
}

entreprise.save_settings(0)
{
short outFileNum;
unsigned char charName[24] = "save.ini";
StandardFileReply outputReply;
char string[235]= [0];
short i;
time another_time;
struct tm *curtime;

if(get_save_file(charName, &outputReply))
{
int 0 = (FspOpenDF((&outputReply).strFile, LSDPerm, &outFileNum)); // Open data file
        idxCursor = GetCursor(stringCursor); // Change the cursor
            SetCursor(*&idxCursor);
        // date
        printf("%s", axftime(curtime));
        write_line(outFileNum, string, 0, 2);
        // control width
        write_line(outFileNum, "CONTROL WIDTH (mb)", 0, 1);
        printf("%4.2f", (float)valve_action_incremen / RAVENS);
        write_line(outFileNum, string, 0, 1);
        write_line(outFileNum, "END CONTROL WIDTH", 0, 2);
        // image width
        write_line(outFileNum, "IMAGE WIDTH (iph)", 0, 1);
        printf("%4.1f", max_image);
        write_line(outFileNum, string, 0, 1);
        write_line(outFileNum, "END IMAGE WIDTH", 0, 2);
        // vi coefficients
        write_line(outFileNum, "V1 IMAGE COEFFICIENTS", 0, 1);
        printf("%4.1f", wtdf[0]h_coeff);
        write_line(outFileNum, string, 0, 1);
        printf("%4.1f", wtdf[0]h_coeff2);
        write_line(outFileNum, string, 0, 1);
        printf("%4.1f", wtdf[0]v_coeff);
        write_line(outFileNum, string, 0, 1);
        printf("%4.1f", wtdf[0]v_coeff2);
        write_line(outFileNum, string, 0, 1);
        write_line(outFileNum, "END VI IMAGE COEFFICIENTS", 0, 2);
    // vi preferences
    write_line(outFileNum, "VI SETTINGS", 0, 1);
        printf("%s", wtdf[0]h_prec);
        write_line(outFileNum, string, 0, 1);
        printf("%s", wtdf[0]v_prec);
        write_line(outFileNum, string, 0, 1);
        printf("%s", wtdf[0]v_prec2);
        write_line(outFileNum, string, 0, 1);
        write_line(outFileNum, "END VI IMAGE COEFFICIENTS", 0, 2);
    // vi settings
    write_line(outFileNum, "VI SETTINGS", 0, 1);
        printf("%s", wtdf[0]h_prec);
        write_line(outFileNum, string, 0, 1);
        printf("%s", wtdf[0]v_prec);
        write_line(outFileNum, string, 0, 1);
        printf("%s", wtdf[0]v_prec2);
        write_line(outFileNum, string, 0, 1);
        write_line(outFileNum, "END VI IMAGE COEFFICIENTS", 0, 2);
FluVoi (NIL, (outputReply).xFile. xRe[Num]);

// ..............................................................

#define EXIT 0

#define EXIT 0

// ..............................................................

#include APPLE_MENU 128
#define WINDOW_TOP 40
#define WINDOW_LEFT 0
#define VD_1_WINDOW_TOP 46
#define VD_1_WINDOW_LEFT 10

typedef control_status {
    Boolean run;
    Boolean show_pixels;
    Boolean show_data;
    Boolean show_shadows;
    Boolean expand;
    Boolean distance_counter;
} status_type = {0,0,0,0,0,0};

#define OVER_SPEED 0x01  // PC0 - overspeed = low
#define DISABLED 0x02    // PC1 - disabled = 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); #include line, error_line);
enum (NO, YES);
enum (OFF, ON);
enum (MANUAL_ON, MANUAL_OFF, AUTO);
#define RAVENS 130    // raven radar detector = 130 pulses
#define TICKS_PER_SEC 60    // macintosh ticks per second
#define PI 3.14159
#define LUTSIZE 256   // brightness/contrast lut size
#define DEF_BRIGHTNESS 150 // default brightness
#define NEW_BRIGHTNESS 150 // default contrast
#define PBLEFT 0      // default contrast
// RECTANGLES
#define RECT_IN_MAJ 130

// CONTROLS
#define ABOUT_BOX_ID 128 // Resource IDs for dialogs
#define IMAGE_MAP_BOX_ID 129
#define LONG_NAP 10L
#define APPEND_MENU 0
#define IN_FRONT 1
#define CHAR_CODE_MASK 255

// a gwindow for every window
typedef struct

// WindowRecord wind;
// PHandle thePlt;
// short paletteCode;
// PWindowHandle thePalette;
// GVWorldPt windGWorld;
// WIndowGWorld, *WindNGWorldPt;

#define LUTSIZE 256
#define MENCOLORVAL 0
#define ORIGIN 1275 // Half of 255, for contrast calculations
#define APPLEPIE 3.14160156525 // Apple's fixed point value of pi

short vd_open_drive(short *refnum);
int openVideo(short *refnum, unsigned long *base);
short vd_set_video_mode(struct CtrIParam *pb, short refnum, int vd_br);
short vd_set_video_mode(struct CtrIParam *pb);
short fill_pb(struct CtrIParam *pb, unsigned long color);
short vd_set_source(struct CtrIParam *pb, int source);
short vd_set_temperature(struct CtrIParam *pb);
short vd_set_max_rect(struct CtrIParam *pb, int *max_width, int *max_height);
short vd_set_max_source_rect(struct CtrIParam *pb, Rect *rect);
short vd_set_rect(struct CtrIParam *pb, Rect *rect, short width, short height, short method);
short vd_set_location(struct CtrIParam *pb, Point topLeft);
short vd_set_video_mode(struct CtrIParam *pb, int mode);
short vd_set_color_level(struct CtrIParam *pb, short *hue);
short vd_set_video_level(struct CtrIParam *pb, short *brightness, short *contrast);
short vd_set_sat_level(struct CtrIParam *pb, short *sat);
short vd_set_segment(struct CtrIParam *pb, int saturation_step);
int grab_frame(short vd_number, short refnum, short mode);
short vd_set_frame_rate(struct CtrIParam *pb, short mode);
int check_frame_grab(short refnum);
short vd_set_subpixel(struct CtrIParam *pb, unsigned char delay);

void makekey(short brightness, short contrast, unsigned char ramp[]);

struct video_digitizer

// ROps.c
#include "RopsDriver.h"
#include <Files.h>
#include <Device.h>
#include <DeviceFuncs.h>
#include <string.h>
#include <math.h>

#define UTABLE 0
#define FIRST_SLOT 0
#define LAST_SLOT 16
#define MAXCOLORVAL 255

int(*video_digitizer)(short vdf[2]);

GDevice **pipe;

// open frame grabber driver
short vd_open_driver(short *refnum)
{
    short err = 0;
    if(!OpenDevice("ps_RasterOps 24MxTV PIPE", refnum))
    {
        *refnum = 0;
        return(err);
    }
    return(0);
}
int openVideo(short *refnum, unsigned long *base)
{
    short error;
    GDevice **dev;
    SpBlock *spb;  // Slot Manager
    short slot, notfound, slotfound;

    notfound = 1;
    for (slot=FIRST SLOT; slot<=LAST SLOT) & & (notfound); slot++) # Exit when first board is found
    {
        spb.spSlot = slot;  
        spb.spID = 0;
        spb.spExDev = 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 24MXIP PIP", refnum);
                    notfound = 0;
                    slotfound = slot;
                    break;
                }
            }
            *base = 0x00000000 | (slotfound << 24);
        }
        if (error) return(error);
        if (notfound) return (notfound);
    }
    dev=GetDeviceList(); /* Get the GDevice */
    while (dev)
    {
        if (notfound) return (notfound);
        dev=GetDeviceList();
        if (slotfound = (for (dev to auxDev); (dev > 0) && (dev & 0x00000001)) pipeId = dev;
        dev=GetNextDevice(dev);
    }
    return(error);
}

// ----- select frame grabber card -------
short vd_select_board(struct CtriParam *pb, short refnum, int vd_br)
{
    short err = 0;
    (*pb).loCredNum = refnum;
    (*pb).loCompletion = NIL;
    (*pb).loVRefNum = 0;
    (*pb).csCode = CONTROL_SELECTIPBOARD;
    (*pb).csParam[0] = vd_br;  // video digitizer order
    err = PBCtrl((PDatabase) pb, false);
    return(err);
}

// reset the video digitizer
short vd_reset(struct CtriParam *pb)
{
    short err = 0;
    (*pb).csCode = CONTROL_PIPRESET;
    err = PBCtrl((PDatabase) pb, false);
    return(err);
}

// set initial video digitizer RAM to black
short fill_fv(struct CtriParam *pb, unsigned long color)
{
    short err = 0;
    long *fillval;
    fillval = &long (*pb).csParam[0];
    *fillval = color;  // Set the grabber RAM to selected color
    (*pb).csCode = 0x0000;  //CONTROL_SET_PIPERASEFRAMEBUFFER;
    err = PBCtrl((PDatabase) pb, false);
    return(err);
}

// set video digitizer source signal
short vd_set_source(struct CtriParam *pb, int source)
{
    short err = 0;
    (*pb).csParam[0] = source;
    (*pb).csCode = CONTROL_SET_PIPSOURCE;
    err = PBCtrl((PDatabase) pb, false);
    return(err);
}

// set Nullbus delay
short vd_set_nullbus(struct CtriParam *pb, unsigned char delay)
{
    short err = 0;
    (*pb).csParam[0] = delay;
    (*pb).csCode = CONTROL_SET_PIPNULLUSEDELAY;
    err = PBCtrl((PDatabase) pb, false);
    return(err);
}

// set framegrabber for full speed (30 frames/sec)
short vd_set_speed(struct CtriParam *pb)
{
    short err = 0;
    (*pb).csCode = CONTROL_SET_PIPSPEED;
    (*pb).csParam[0] = 0;  // PIP
    err = PBCtrl((PDatabase) pb, false);
    return(err);
}

// find maximum video rectangle supported (max. width and height)
short vd_find_max_source_rect(struct CtriParam *pb, Rect *rect)
{
    short err = 0;
    (*pb).csCode = STATUS_GET_PIPMAXRECT;
    if (err = PBCtrl((PDatabase) pb, false))
        return(err);
    (*rect).top = (*pb).csParam[0];  // top
    (*rect).left = (*pb).csParam[1];  // left
    return(err);
}

Copyright 2011, AHMCT Research Center, UC Davis
short vd_set_sct_level(CntlParam *pb, short *sat)
|
short err = 0;
(*pb).csCode = CONTROL_SET_SATURATION;
(*pb).csParam[0] = *sat;
err = PBCtrl((P工作任务Param P工作任务 Param); false);
return(err);
|
short vd_fld_max_test(struct CntlParam *pb, int *max_width, int *max_height)
|
short err = 0;
(*pb).csCode = STATUS_GET_MAXDESTSIZE;
err = PBSStatus((P工作任务Param P工作任务 Param)); false);
*max_height = (*pb).csHeight();
*max_width = (*pb).csParam[0];
return(err);
|
short vd_set_rect(struct CntlParam *pb, Rect vrect, short width, short height, short method)
|
short err = 0;
(*pb).csCode = CONTROL_SET_PIPDIGITIZEDVIDEORECT;
(*pb).csParam[0] = (*vrect).top;
// sampling method flag
err = PBCtrl((P工作任务Param P工作任务 Param); false);
return(err);
|
short vd_set_location(struct CntlParam *pb, Point topleft)
|
short err = 0;
(*pb).csCode = CONTROL_SET_PIPSURFACEPOS;
(*pb).csParam[0] = topleft.v;
// top left of frame buffer on screen
err = PBCtrl((P工作任务Param P工作任务 Param); false);
return(err);
|
short vd_set_digitize_mode(struct CntlParam *pb, int mode)
|
short err = 0;
(*pb).csCode = CONTROL_SET_PIPONESHOT;
(*pb).csParam[0] = mode;
err = PBCtrl((P工作任务Param P工作任务 Param); false);
return(err);
|
|
short vd_set_hue_level(CntlParam *pb, short *hue)
|
short err = 0;
(*pb).csCode = CONTROL_SET_HUE;
(*pb).csParam[0] = *hue;
err = PBCtrl((P工作任务Param P工作任务 Param); false);
return(err);
|
short vd_set_sat_level(CntlParam *pb, short *sat)
|
short err = 0;
(*pb).csCode = CONTROL_SET_SATURATION;
(*pb).csParam[0] = *sat;
err = PBCtrl((P工作任务Param P工作任务 Param); false);
return(err);
|
// frame-grabber brightness and contrast lui control
short vd_set_video_level(CntlParam *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
fp = fopen("ui_frame_grabber.lut", "r");
if (fp != NULL) {
fclose(fp);
} else {
*brightness = DEFIBRIGHTNESS;
*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
(*pb).csParam[0] = *brightness;
err = PBCtrl((P工作任务Param P工作任务 Param); 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.
Brightens 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 linear ramp (default).
|
void makebclut(short brightness, short contrast, unsigned char *ramp)
|
int k;
long val;
long multfactor, originfactor;
Fract a, c, s, c0, c1;
Fixed temp;
Fixed pi;

Copyright 2011, AHMCT Research Center, UC Davis
Copyright © 1995 by University of California Davis
long double apple1 = 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 / 256);
pi = X2Fix(apple1);
sine = FixSin(FixMul(temp, pi));
 cosine = FixCos(FixMul(pi, sine));
multfactor = FixDiv(Fix2Fix(sine), Fix2Fix(cosine)); /* tangent
originfactor = FixMul(FixRatio(1, 1) - multfactor, X2Fix(origin));

for (i = 0; i < LUTSIZE; ++i)
{ val = Fix2Lin(FixMul(FixRatio(1, 1), multfactor) + originfactor);
val = (brightness - 128);
if (val < MINCOLORVAL) val = MINCOLORVAL;
else if (val > MAXCOLORVAL) val = MAXCOLORVAL;
ramp[i] = val;
}

int check_frame_grab(short refnum) {
  CntiParam pb;
  pb.ioCRefNum = refnum;
  pb.ioCompletion = nil;
  pb.ioVRefNum = 0;
  pb.csCode = STATUS_GET_PIPACTIVE;
  PBstatus((ParmBkPt*pb)->false);
  return(pb.csParam[0]); // 1 active, 0 not active
}

......
#include <string.h>
#include <stdio.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 0x000000ff
#define GREEN 0x0000ffff
#define BLUE 0x000000ff
#define DARKGRAY 0x00333333
#define MEDIUMGRAY 0x00666666
#define LIGHTGRAY 0x00999999
#define ONE_FILE_TYPE: 0
#define NEW_LINE 0x0D
#define TAB 0x09

Rect status_rect, info_rect, error_rect;

struct circle {
    Point center;
    short radius;
};

CType g_fileCreator = { MPCC };
CType g_fileType = { TEXT };
CurHandle g_fileHandle;

void pause(float pause_time);
void color_screen(unsigned long color);
void my_draw_point(short x, short y, unsigned long color, char skip);
void my_draw_line(short x1, short y1, short x2, short y2, unsigned long color);
void my_draw_frame(Rect *fr, char *string);
void my_print(char info_type, char *message, int value);
float my_round(float number);
void my_draw_rect(Rect rect, unsigned long color);
//void AskFile(void);
short get_save_file(unsigned char fileName[], StandardFileReply *outputReply);
void report_error(OSErr errCode);
void write_list(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 tabs(short outFileRefNum, short tab_count);
void my_draw_circle(circle *circle, unsigned long color);
void my_draw_round(circle *circle, unsigned long color);
void my_draw_line(Point p1, Point p2, unsigned long color);
void my_draw_v_line(Rect level_rect, short level, short min, short max);
void my_draw_h_line(Rect level_rect, short level, short min, short max);
void my_draw_v_line(Rect level_rect, short level, short min, short max);
void my_draw_h_line(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 max, min, maxv, mindv;
    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
    min = 20;
    max = 812; // horz screen pixels
    maxv = 608; // vert screen pixels
    hpxl_step = hstep * 0x00000004; // horz memory increment
    vpxl_step = 0x0000000400; // vert memory increment
    pxl_addr = base + (v + vpxl_step + mindv * hpxl_step); // first pixel
    for (v = mindv; v < maxv; v++) { // select rows
        for (h = min; h < max; h++) { // select columns
            pxl_color = (unsigned long *) pxl_addr; // get pixel location
            pxl_addr += (unsigned long) hpxl_step; // move to next pixel column
        }
    pxl_addr = base + (v * vpxl_step + mindv * hpxl_step); // next pixel row
    }
}
void my_draw_point(short x, short y, unsigned long color)
{
    unsigned long hpxl_step, vpxl_step;
    unsigned char *phtl_color;
    unsigned long *phtl_color addr;
    base = 0[0];
    hpxl_step = 0x000000000;
    vpxl_step = 0x000000000;
    phtl_color = (unsigned long *) base + x * hpxl_step + y * vpxl_step;
    phtl_color = &color;
    my_draw_point(x, y, TOP_FR);
    right --;
}

// frame, top
for(x = left; x < right; x++)
    for(y = top; y <= bottom; y++)
        my_draw_point(x, y, RIGHT_FR);
}

// frame, bottom
for(x = left; x < right; x++)
    for(y = top; y <= bottom; y++)
        my_draw_point(x, y, BOTTOM_FR);
}

// frame, right
for(x = left; x < right; x++)
    for(y = top; y <= bottom; y++)
        my_draw_point(x, y, LEFT_FR);

// frame, left
for(x = left; x < right; x++)
    for(y = top; y <= bottom; y++)
        my_draw_point(x, y, FR_LEFT_TH);

// frame, up
for(x = left; x < right; x++)
    for(y = top; y <= bottom; y++)
        my_draw_point(x, y, FR_UP_TH);

// frame, down
for(x = left; x < right; x++)
    for(y = top; y <= bottom; y++)
        my_draw_point(x, y, FR_DOWN_TH);

// frame, up-right
for(x = left; x < right; x++)
    for(y = top; y <= bottom; y++)
        my_draw_point(x, y, UP_RIGHT_TH);

// frame, up-left
for(x = left; x < right; x++)
    for(y = top; y <= bottom; y++)
        my_draw_point(x, y, UP_LEFT_TH);

// frame, down-right
for(x = left; x < right; x++)
    for(y = top; y <= bottom; y++)
        my_draw_point(x, y, DOWN_RIGHT_TH);

// frame, down-left
for(x = left; x < right; x++)
    for(y = top; y <= bottom; y++)
        my_draw_point(x, y, DOWN_LEFT_TH);
```c
// my_draw_circle(circle *circle, unsigned long color)
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.y + shortpow(((double)(x * r) - (double)(x * x)), 0.5);
    my_draw_point(x + (*circle).center.x, y, color);
    y = (*circle).center.y - shortpow(((double)(x * r) - (double)(x * x)), 0.5);
    my_draw_point(x + (*circle).center.x, y, color);
}

// my_fill_circle(circle *circle, unsigned long 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.y + shortpow(((double)(r * r) - (double)(x * x)), 0.5);
    y_top = (*circle).center.y - shortpow(((double)(r * r) - (double)(x * x)), 0.5);
    for(y = y_top; y > y_bottom; y++)
        my_draw_point(x + (*circle).center.x, y, color);
}

// my_draw_line(Point p1, Point p2, unsigned long color)
void my_draw_line(Point p1, Point p2, unsigned long color)
{
    short x, y, m, b;
    if(p2.y - p1.y) / (p2.x - p1.x) < p1.y)
        m = (p2.y - p1.y) / (p2.x - p1.x);
    else if(p1.y < p2.y)
        m = (p2.y - p1.y) / (p2.x - p1.x);
    else if(p1.y < p2.y)
        m = (p2.y - p1.y) / (p2.x - p1.x);
    else
        m = (p2.y - p1.y) / (p2.x - p1.x);
    for(x = p1.x; x < p2.x; x++)
        y = m * x + b;
    my_draw_point(x, y, color);
}

float height = 0;
for first_top = level.rect.top;

if(level <= max && level >= min)
    height = ((float)level / (max - min)) * (level.rect.bottom - level.rect.top);
level.rect.top = level.rect.bottom - (2*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);

void my_draw_v_line(Rec rect, short level, short min, short max)
{
    int level_rect_mid, left_side, right_side;
    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);
        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_right;
    ```
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_r_tilelevel(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))
  {
    top_side = level_rect_mid;
    bottom_side = level_rect_mid + ((float) level / (max - min)) * 0.5 * (level_rect.bottom - 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 %d", message, value);
  else
  sprintf(print_string, "#s", message);

  switch(info_type)
  {
  // case status_line:
  case status_line:
    EraseRect(&status_rect);
    MoveTo(status_rect.left, status_rect.top + 12);
    drawstring(print_string);
    break;
  case info_line:
    EraseRect(&info_rect);
    MoveToInfoRectLeft(info_rect_left, info_rect.top + 12);
    drawstring(print_string);
    break;
  case error_line:
    EraseRect(&error_rect);
    MoveToErrorRectLeft(error_rect_left, error_rect.top + 12);
    drawstring(print_string);
    SysBeep(60);
    break;
  }

  //........................................................................
  // float my_round(float number)
  {
    if((int) number < number + 0.5)
      number = (int) number;
    else
      number = (int) number + 1;
  }

  //........................................................................
  // Obtain save file name
  short __save_file(unsigned char fileName[], StandardReply *outputReply)
  {
    short inFileRefNum;
    OSErr fileError;
    short oldVol;
    SFTypeList textType = "TEXT";

    Open the output file
    StandardPutFile("psSave settings in ", fileName, outputReply);
    if(0 != outputReply->file)
    {
      SetVol NIL, ("outputReply"); // Make the destination volume current
      fileError = FSpCreate(0, outputReply, sFile, gfileCreator, gfileType, smSystemScript);
      switch(fileError)
      {
      case noErr:
        return 1;
      case dupFNErr:
        // File already exists, wipe it out
        if((fileError == FSExists) & (outputReply) == noErr)
        {
          if(fileError == FSpCreate(0, outputReply, sFile, gfileCreator, gfileType, smSystemScript)) == noErr)
          {
            report_error(fileError);
            FSClose(inFileRefNum);
            SetVol(NIL, oldVol);
            return 0;
          } // end if
          return 1;
        } // end == noErr
        else
        {
          report_error(fileError);
          FSClose(inFileRefNum);
          SetVol(NIL, oldVol);
          return 0;
        } // end else
        break; // end case dupFNErr
        default:
          report_error(Nil fileError);
        } // end switch
      } // end if
    } // end if
  } // end function

Copyright © 1995 by University of California Davis
FSWrite(outFileRefNum, &amount, string);

void func(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 func(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);
    }
}

// Increment the tab count
short inc_tab_count(void)
{
    short tab_count;
    return 1;
}

// Increment the line count
short inc_line_count(void)
{
    short line_count;
    return 1;
}

void func(short outFileRefNum)
{
    FSWrite(outFileRefNum, &amount, string);
}

void func(short outFileRefNum)
{
    FSWrite(outFileRefNum, &amount, &control);
}

// Constructor
struct Box
{
    short x, y, width, height;
}

// Destructor
void Box::~Box()
{
    delete[] buffer;
}

int main()
{
    struct Box box;
    box.x = 10;
    box.y = 20;
    box.width = 50;
    box.height = 30;
    return 0;
}

#include "gauss.h"

unsigned long *tput1, **tbuf1;
unsigned long *tput2, **tbuf2;

extern Rect *img_inited_rect;
extern EVENT_RECORD *eEvent;
extern Rect *img_hpixel_rect;
extern Rect *img_vpixel_rect;
extern Rect *img_wpixel_rect;
extern Rect *img_spixel_rect;
extern Rect *img_lpixel_rect;
extern Rect *img_rpixel_rect;
extern Rect *img_tpixel_rect;
extern Rect *img_bpixel_rect;
extern Rect *img_mpixel_rect;
extern Rect *img_mpixel_rect;
extern Rect *img_trpixel_rect;
extern Rect *img_blpixel_rect;
extern Rect *img_mlpixel_rect;
extern Rect *img_brpixel_rect;

void showlut(unsigned long base);
int getlut();
void do_img_map(unsigned short width);
extern short mouse_x,(Rect c_rect, Pixel click);
extern void do_rect(Rect rect, string char align);
extern void gauss_solve doubly;[double i][double out]);

#define region of interest (roi)

class roi {
    private:
        unsigned long ps1_addr, hpixel_step, vpixel_step;
        short minh, maxh, minv, maxv, minn, maxn;
        short sel;
    public:
        void set_roi(char *tstring, float set_width, float *band_adj, unsigned long base, float voi[setctl];
        int analyze_roi(int offset, int show_pixel);
        void add_roi(Rect *shif, unsigned long base);
        short find_max_pixel(void);
        short find_max_m_pixel(void);
        short find_max_v_pixel(void);
        void add_roi(Rect *nask);
        float roi_xtlside(void);
        float roi_ytleftside(void);
        float roi_brightside(void);
        float roi_entside(void);
        float roi_lshortside(void);
        float roi_bshortside(void);
        void roi_entshortside(float float value, unsigned long base, float *band_adj);
        void roi_yentshortside(float float value);
        void roi_ylshortside(int int value);
        void roi_bshortside(int int value);
        void roi_vshortside(unsigned long base);
        void roi_entshortside(unsigned long control_increment);
        void shift_band(float roi_voffset);
        void make_big_rect(Rect *rect);
    
    #define analyze region in video image (specified by borders, pixel density, noise allowance)

    void roi::analyze_roi(int offset, int show_pixel) {
        int register int h, v, pixel_count;
        unsigned long ps1_addr, pixel_value;
        unsigned short new_pxl_addr, start_pxl;
        start_pxl = ps1_addr + offset * vpixel_step;
        for(v = minv; +maxv; ++vstep) { // vertical scan
            new_pxl_addr = v * vpixel_step + start_pxl; // left starting pixel
            for(h = minh; +maxh; +steph) {

            } // horizontal scan
    
    #define convert roi's from meters to pixels

    void roi::set_roi(char *tstring, float set_width, float *band_adj, unsigned long base, float voi[setctl]);
Copyright 2011, AHMCT Research Center, UC Davis

Copyright © 1995 by University of California Davis

Copyright 2011, AHMCT Research Center, UC Davis
void roi-edit_roi_farside(float value)
|
| short vd_number = 0;
| maxhm = value;
| max = 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)
| |
| (float)control increment / RAVENS;
| max = my_round(RAVENS * (400.77 + 113.9 * minhm) + (8.2 * minhm * minhm)); // bottom of roi
| |
| void roi-make_big_zone(Rect *rect)
| |
| short vd_number = 0;
| minhm = *(rect).left;
| maxhm = *(rect).top;
| max = *(rect).right;
| max = *(rect).bottom;
| hstep = 1;
| vstep = 1;
| noise = 1;
| hpxl_step = hstep * 0x00000004;
| vpxl_step = vstep * 0x00000000;
| psl_addr = vd[vd_number].base + (minhm * hpxl_step + vstep);
| |
| void showlut(unsigned long base)
| |
| short *psl_value, lut_color = 0x00000000, hpxl_step;
| short *min, max, vstep, mish, maxhm, hflag = 0;
| unsigned long vpxl_step, psl_addr, h, hpxl_step;
| EventRecord *event;
| char key;
| short delay_loop;
| WindowPtr LUT_Window, current_window;
| Rect LUT_rect;
| current_window = FronWindow();
| HidewIndow(current_window);
| SetRect(&LUT_rect, 50, 50, 620, 620);
| LUT_Window = NewCWWindow(NULL, &LUT_rect, "CLUT", true, 0x1000100,
| WindowProc(NULL), false, NULL);
| minv = 20;
| maxv = minv+256;
| vstep = 1;
| mish = 20;
| maxhm = mish+256;
| hstep = 1;
| hpxl_step = hstep * 0x00000004; // horz memory increment
| vpxl_step = vstep * 0x00000000; // vert memory increment
| psl_addr = base + (minv * vpxl_step + mish * hpxl_step); // first pixel
| while(lut_color <= 0x00000000)
| |
| for(v=minv; v<maxv; v+=vstep) // select row
| |
| for(h=minhm; h<maxhm; h+=hstep) // select column
| |
| psl_value = (unsigned long *) psl_start; // get pixel value address
| if(!!!((lut_color & 0x0000000000000000) == (lut_color & 0x0000000000000000)))
| psl_value = 0x00000000;
| else
| psl_value = lut_color;
| (lut_color = lut_color & (lut_color - 1)) & ((lut_color = lut_color & (lut_color - 1))& (lut_color & 0x0000000000000000))
| psl_value = 0x00000000;
| psl_addr = (unsigned long)vpxl_step; // move to next pixel column
| lut_color++;
| |
| psl_addr = base + (v * vpxl_step + mish * hpxl_step); // next pixel row
| |
| mish++; // provide "3-D" offset
| minv++;
| maxv++;
| maxhm++;
| if(lut_color >= 0x00000000) // pause if end of color space reached
| delay_loop = 1;
| do
| |
| GetNextEvent(everyEvent, event);
| switch (even weap)
| |
| case keyDown:
| // keyboard input
| key = event->message & charCodeMask;
| switch (key)
| |
| case wp:
| |
| delay_loop = (delay_loop + 10) / 10;
| if(delay_loop
| |
| psl_addr = base + (minv * vpxl_step + mish * hpxl_step); // first pixel
| vstep = 1;
| // select row
| for(h=minhm; h<maxhm; h+=hstep) // select column
| |
| psl_value = (unsigned long *) psl_addr; // get pixel value address
| Copyright 2011, AHMCT Research Center, UC Davis
| Copyright © 1995 by University of California Davis
"pixel_value = 0x00000000;  // move to next pixel column

pixel_addr = base + (minv * vpxl_step + msh * hpxl_step);  // first pixel
horizontal;  // select row
for(v=eminv; v<maxv; v+=vstep)
  // select columns
    "pixel_value = (unsigned long *) pixel_addr;  // get pixel value address
    "pixel_value = 0x00000000;
    "pixel_addr = base + (v * vpxl_step + msh * hpxl_step);  // next pixel row
  break;
  case 'c':
    SysBeep(1);
    get_to_clip(LUT_rect.left, LUT_rect.top, LUT_rect.right, LUT_rect.left, LUT_rect.bottom);
    get_to_clip(LUT_rect.left, LUT_rect.top, LUT_rect.left+400, LUT_rect.top+400);
    break;
    }
  while (delay_loop);
  pixel_addr = base + (minv * vpxl_step + msh * hpxl_step);  // start of next rectangle
    }
  DisposeWindow(LUT_Window);
  ShowWindow(current_window);
  }
  } // end of lut display
  } // end of get_lut
  }
  FILE *fid;
  int p1 = nil; int h1 = nil;
  int p2 = nil; int h2 = 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 );
      lutp = (unsigned long *) *luth1;
      fread(lutp1, sizeof(long), 524288, fid);
    }
    "SysBeep(1);
    "myprint(status_line, "Standard Look-up table file error", 0);
    "pause(5.0);
    }
  fclose(fid);
  return(0);  // all o.k. - return 0
  }
  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);
    }
    "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 heuristic[10] = {0.0}, v-metric[10] = {0.0};
  unsigned long color = 0x00000000;
  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.x = 0;
      • mark[i][j].center.y = 0;
      • cursor_pos.radius = 3;
    hpxl_step = 0x00000000;  // horz memory increment
vpx1_step = 0x00000000; // vert memory increment

first_vd_right = vd[vd_number].image_rect.right;
vfd = 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);

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

spritef(string, "set number: %d", set);
myprintf(status_line, string, 0);

while(mouse_in(image_map_finished_rect, mouse_down_rect) && set < 10) |

WaitNextEvent(everyEvent, &everyEvent, LONG_NAP, NO_CURSOR);
if(everyEvent.what == mouseDown)
GetMouse(&mouse_down_loc);
if(mouse_in(vd[vd_number].disp_rect, &everyEvent.where)) |
    x = &everyEvent.where.x;
y = &everyEvent.where.y;
hideCursor();
grab_frame_t(1, vd[vd_number].refsum, WAIT); // first slot with vd
if(t == 0)
    mark[set][0].center.h = x;
mark[set][1].center.v = y;
cursor_pos.center.h = x;
cursor_pos.center.v = y;
my_fill_circle(&cursor_pos, GREEN);
for(i = 0; i < set; i++) |
    my_fill_circle(&mark[i][0], BLUE);
    my_fill_circle(&mark[i][1], BLUE);
    my_draw_line(mark[i][0], mark[i][1], center, BLUE);
my_fill_circle(&mark[set][0], RED);
if(set == 1)
    my_fill_circle(&mark[set][1], RED);
    my_draw_line(mark[set][0], mark[set][1], center, RED);
sprintff(string, "%d", i);
myprintf(status_line, string, 0);
sprintff("> %d", 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);
sprintff("%d", 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(everyEvent.what == mouseDown)

if(set == 1)

hset = mark[set][0].center.h - vd[vd_number].disp_rect.left;
set = mark[set][0].center.x - mark[set][0].center.x;
if(vd[vd_number].h_coeff1 * vd[vd_number].h_coeff1 + 4 * vd[vd_number].h_coeff2 * (float)hset > 0)
    hmeter[set] = i - vd[vd_number].h_coeff1 + sqrt(vd[vd_number].h_coeff2 * (float)hset) / (2 * vd[vd_number].h_coeff2);
else
    hmeter[set] = 3.0;
vmeter[set] = (float)vmeter[set] / (vd[vd_number].v_coeff0 + vd[vd_number].v_coeff1 * hmeter[set] + vd[vd_number].v_coeff2 * hmeter[set]) * hmeter[set];

set++; | 

sprintff(string, "set number: %d", set);
myprintf(status_line, string, 0);
do |

if(mouse_in(next_image_map, &everyEvent.where)) |

set++; |

if(everyEvent.what == mouseDown)

if(mouse_in(more_hmeter_rect, &everyEvent.where)) |

hmeter[set] = 0.01;
sprintff(string, "h meter: %f", 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, &everyEvent.where)) |

hmeter[set] = 0.01;
sprintff(string, "h meter: %f", hmeter[set]);
my_fill_rect(&image_map.hmeter_rect, WHITE, 1);
do_text_rect(image_map, hv_meter_rect, string, 0);

if(mouseInOut(mouse_vmeter_rect, gEvent.where))
{
  vmeter[set] = 0.01;
  sprintf(string, "v meter: %.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: %.2f", vmeter[set]);
  my_fill_rect(&image_map, vmeter_rect, WHITE, 1);
  do_text_rect(image_map, vmeter_rect, string, 0);
}

if(SHITdown)
{
  pause(0.1);
  while(SHITdown);
}

grab_frame(1, vd[vd_number].refsum, DONTWAIT);

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

  v[vd_number].v_coef1 = c[1];
  v[vd_number].v_coef2 = c[2];

  sprintf(string, "v: %.2f, %.2f, %.2f", c[0], c[1], c[2]);
  myprintf(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] += hv_meter[i];
}
MenuHandle gmyMenu[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)
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
void set_status(char *string)
{
    // start with valve in auto mode
    status = AUTO;
}

// calculate valve control signal based on plants detected

Boolean calculate_valve_signal(unsigned long plants[], float speed, char base_rate, int valve_action_increment)
{
    // declare variables
    short band, back_band, front_plant, first_band, last_band;
    unsigned long valve_signal[10];
    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_bill) + ((dist - f_lead - (t0f_adj + d_lead) * speed)) / valve_action_increment;
    front_plant = (base_bill) + ((dist - f_lead - (t0f_adj + d_lead) * speed)) / valve_action_increment;

    if (status.expand)
    {
        back_plant = front_plant;
        first_band = expand_last;
        last_band = expand_first;
    }
}

// watch for over-run of data
if(front_plant < data_count && front_plant > 31 || back_plant < data_count || back_plant > 31)
{
    do_overrun_warning(22);
    return 1;
}

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

while(last_band >= data_count)
{
    // can't look further out than the last one
    last_band = data_count;
}

// 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] <= (11-back_plant) || (11-back_plant) <= front_plant);

    // display the data if desired
    if(status.show_data)
    {
        show_signal(band, last_front_plant[band], last_back_plant[band], 1.0, 0.0000000000000000);
        show_signal(band, front_plant-1, back_plant-1, 1.0, 0.0000000000000000);
        show_signal(band, last_back_plant[band], last_front_plant[band], 1.0, 0.0000000000000000);
    }
}

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

//........................................................................
float valves::get_dist()
{
    return dist / RAVENS;
}

//........................................................................
float valves::get_tof()
{
    return t0f_adj / TICKS_PER_SEC;
}
```c
float valvec::get_f_lead()
{
    return f_lead / RAVENS;
}
float valvec::get_f_lag()
{
    return f_lag / RAVENS;
}
float valvec::get_d_lead()
{
    return d_lead / TICKS_PER_SEC;
}
float valvec::get_d_lag()
{
    return d_lag / TICKS_PER_SEC;
}
short valvec::get_band_in()
{
    return band_in;
}
short valvec::get_band_out()
{
    return band_out;
}
float valvec::get_expand_norm()
{
    return expand_norm;
}
float valvec::get_expand_tang()
{
    return expand_tang;
}
short valvec::get_status()
{
    return valve_status;
}
void valvec::edit_f_lead(float new_f_lead)
{
    f_lead = new_f_lead * RAVENS;
}
void valvec::edit_f_lag(float new_f_lag)
{
    f_lag = new_f_lag * RAVENS;
}
void valvec::edit_d_lead(float new_d_lead)
{
    d_lead = new_d_lead * TICKS_PER_SEC;
}
void valvec::edit_d_lag(float new_d_lag)
{
    d_lag = new_d_lag * TICKS_PER_SEC;
}
void valvec::edit_band_in(short new_band_in)
{
    band_in = new_band_in;
}
void valvec::edit_band_out(short new_band_out)
{
    band_out = new_band_out;
}
void valvec::edit_expand_norm(float new_coef)
{
    expand_norm = new_coef;
}
void valvec::edit_expand_tang(float new_coef)
{
    expand_tang = new_coef;
}
void valvec::set_status(short control_status)
{
    valve_status = control_status;
}
```

---

// toao_display.h

// This header file contains the functions used for displaying the target activated
// offset sprayer (TASO) control information

// Written by Chris Tauer
// 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 0x00000000
#define GREEN 0x00000000
#define BLUE 0x00000000
#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 ro il 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, KI0, wind speed rect[11];

// overspeed rectangles
Rect overspeed_rect[0], overspeed_rect[1], overspeed_rect[2], overspeed_rect[3];

// video digitizer adjustment rectangles
Rect vd_adjust_rect, vd contrast_rect, vd brightness_rect, finished_rect,
vd more contrast_rect, vd less contrast_rect, vd more brightness_rect,
vd less brightness_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, t_video_rect, composite_rect,
split video_rect, interleaved video_rect, vd mode_rect,
vd adj finished_rect, vd use lat rect, apply lat rect;

// ro il adjustment rectangles
Rect band rect[MAX_BAND+1], near side rect, far side rect, hor rect, vert rect,
nose 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, image height rect, more image rect;

// valve data adjustment rectangles
Rect vd adjusting[MAX_BAND+1], volc utp[MAX_BAND+1], volc dwn[MAX_BAND+1], dist rect,
tof rect, f lead rect, d lag rect, d lead rect, d lag rect, band in rect,
band out rect, delete valve rect, add valve rect, volc finished rect,
vdc utp, wind norm coord rect, wind long coord rect;

// valve control rectangles
Rect valve on rect[MAX_BAND+1], valve off rect[MAX_BAND+1], valve auto rect[MAX_BAND+1],
valve control rect, on all rect, all auto rect, all off rect;

// image map rectangles
Rect image map finished_rect, image map layer pixel rect, image map top pixel rect,
image map bottom pixel rect, image map top layer rect, image map bottom layer rect,
image map layer rect, image map top layer rect, image map bottom layer rect;

// globals
extern unsigned long valve_action_increment;
extern short valve_count;
extern short max_image_v;

// function declarations
void initialize_display(short band_count, short valve_count);
void draw_mainscreen(void);
void draw_data_area(bool status);
void draw_video_area(bool status);
void do_overspeed_warning(char warning type);
void draw_over_speed_area(bool status);
void draw_speed_area(bool status);
void draw_v_d_adjust_area(bool status);
void draw_oi_adjust_area(bool status, short band_count, roi zone[MAX_SET+1][MAX_BAND+1]);
void do test rect(Rec rect, char *string, char align);
void draw up button(Rec rect, unsigned long color);
void draw down button(Rec rect, unsigned long color);
void draw control buttons(Rec info rect, short width, short height);
void draw ce(Rect c rect, Rec h rect, char *strings);
void draw valve_adjust_area(bool status, short valve_count, valves value[MAX_BAND+1]);
void display valve control(short v, short new status);
void draw control(Rec status);
void draw image map screen(bool status);
void draw depressed button(Rec rect);
void make rect(Rec rect, pixels top left, short width, short height);
extern void my draw_5 level(Rec 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;
    short rect[16];
    // text lines
    SetPen(&topleft, 10, 597);
    width = 300;
    height = 16;
    make rec(&status rect, topleft, width, height);
    // data area
    SetPen(&topleft, 670, VD_1 WINDOW TOP + 204);
    width = band count * (8 + 1);
    height = 32 * (8 / 16);
    make rec(&data rect, topleft, width, height);
    // video digitizer adjustment
    SetPen(&topleft, 670, VD_1 WINDOW TOP);
    width = 160;
    height = 320;
    make rec(&vd adjust rect, topleft, width, height);
    // contrast
    SetPen(&topleft, vd adjust rect.left + 15, vd adjust rect.top + 10);
    width = 10;
    height = 250;
    make rec(&vd contrast rect, topleft, width, height);
    SetPen(&topleft, vd contrast rect.left - 10, vd contrast rect.bottom + 15);
    width = 10;
    height = 16;
}
control_width_rect.bottom;
SetRect(&image_width_rect, a.h - 100, a.v + 1 * 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 + 1 * 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(&ori_title, a.h + height, noise_rect.right, a.v);
| SetRect(&ori_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, 
| add_side_rect.left + 13, roi_finished_rect.bottom);
| SetRect(&delete_roi_rect, add_roi_rect.left, add_roi_rect.top + height, 
| add_roi_rect.right, add_roi_rect.bottom + 2 * height);

// value adjustment
a.h = 110;
a.v = 150;
width = 70;
height = 20;
b.v = a.v + value_count * height;
SetRect(&lwi_rect, a.h + width, a.v + height, a.h + width, b.v);
SetRect(&lwi_rect, a.h + width, a.v + height, a.h + 2 * width, b.v);
SetRect(&lwi_rect, a.h + 2 * width, a.v + height, a.h + 3 * width, b.v);
SetRect(&lwi_rect, a.h + 3 * width, a.v + height, a.h + 4 * width, b.v);
SetRect(&lwi_rect, a.h + 4 * width, a.v + height, a.h + 5 * width, b.v);
SetRect(&lwi_rect, a.h + 5 * width, a.v + height, a.h + 6 * width, b.v);
SetRect(&lwi_rect, a.h + 6 * width, a.v + height, a.h + 7 * width, b.v);
SetRect(&lwi_rect, a.h + 7 * width, a.v + height, a.h + 8 * width, b.v);
SetRect(&lwi_rect, a.h + 8 * width, a.v + height, a.h + 9 * width, b.v);
SetRect(&lwi_rect, a.h + 9 * width, a.v + height, a.h + 10 * width, b.v);
for(i=0; i<le_value_count; i++)
| SetRect(&lwi_rect[i], a.h - 100, a.v + 1 * height, wind_tag_coef_rect.right, 
| a.v + (i+1) * height);
| SetRect(&lwi_up[i], wind_tag_coef_rect.left, wind_tag_coef_rect.top, 
| wind_tag_coef_rect.right, wind_tag_coef_rect.top + height/2);
| SetRect(&lwi_down[i], wind_tag_coef_rect.left, wind_tag_coef_rect.bottom - height/2 + 1, 
| wind_tag_coef_rect.right, wind_tag_coef_rect.bottom);
| SetRect(&ori_title, a.h + height, noise_rect.right, a.v);
SetRect(&ori_finished_rect, 400, wind_tag_coef_rect.bottom + 30, 450, 
wind_tag_coef_rect.bottom + 50);
SetRect(&add_valve_rect, wind_tag_coef_rect[0].left, ori_finished_rect.top, 
ori_finished_rect.left + 12, ori_finished_rect.bottom);
SetRect(&delete_valve_rect, add_valve_rect.left, add_valve_rect.top + 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_VALUE; 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(&valve_auto_rect[MAX_VALUE-1].right + 15, 
valve_auto_rect[MAX_VALUE-1].top, valve_auto_rect[MAX_VALUE-1].right + 15 * width, 
valve_auto_rect[MAX_VALUE-1].bottom);
| SetRect(&valve_on_rect[MAX_VALUE-1].right + 15, 
valve_on_rect[MAX_VALUE-1].top, valve_on_rect[MAX_VALUE-1].right + 15 * width, 
valve_on_rect[MAX_VALUE-1].bottom);
| SetRect(&valve_off_rect[MAX_VALUE-1].right + 15, 
valve_off_rect[MAX_VALUE-1].top, valve_off_rect[MAX_VALUE-1].right + 15 * width, 
valve_off_rect[MAX_VALUE-1].bottom);
SetRect(&valve_control_rect, a.h, a.v, a.h + MAX_VALUE * width + 15, 
a.v + 3 * height);

// program control
a.h = all_auto_rect.right + 15;
a.v = valve_rect.top;
width = 40;
height = 32;
SetRect(&rdar_rect, a.h, a.v, a.h + width, a.v + height);
SetRect(&rds_rect, a.h + 1 * width + 1, a.v + 2 * width + 1, a.v + height);
SetRect(&rdx_rect, a.h + 2 * width + 1, a.v + 3 * width + 1, a.v + height);
SetRect(&rdy_rect, a.h + 3 * width + 1, a.v + 4 * width + 1, a.v + height);
SetRect(&rdz_rect, a.h + 4 * width + 1, a.v + 5 * width + 1, a.v + height);
SetRect(&rdw_rect, a.h + 5 * width + 1, a.v + 6 * width + 1, a.v + height);
SetRect(&rdn_rect, a.h + 6 * width + 1, a.v + 7 * width + 1, a.v + height);
SetRect(&rdl_rect, a.h + 7 * width + 1, a.v + 8 * width + 1, a.v + height);
SetRect(&rd0_rect, a.h + 8 * width + 1, a.v + 9 * width + 1, a.v + height);
SetRect(&rd1_rect, a.h + 9 * width + 1, a.v + 10 * width + 1, a.v + height);

for(i=0; i<le_value_count; i++)
| SetRect(&rdar_rect[i], a.h - 100, a.v + 1 * height, wind_tag_coef_rect.right, 
| a.v + (i+1) * height);
| SetRect(&rdar_up[i], wind_tag_coef_rect.left, wind_tag_coef_rect.top, 
| wind_tag_coef_rect.right, wind_tag_coef_rect.top + height/2);
| SetRect(&rdar_down[i], wind_tag_coef_rect.left, wind_tag_coef_rect.bottom - height/2 + 1, 
| wind_tag_coef_rect.right, wind_tag_coef_rect.bottom);
| SetRect(&ori_title, a.h + height, noise_rect.right, a.v);
SetRect(&ori_finished_rect, 400, wind_tag_coef_rect.bottom + 30, 450, 
wind_tag_coef_rect.bottom + 50);
SetRect(&add_valve_rect, wind_tag_coef_rect[0].left, ori_finished_rect.top, 
ori_finished_rect.left + 12, ori_finished_rect.bottom);
SetRect(&delete_valve_rect, add_valve_rect.left, add_valve_rect.top + height, 
add_valve_rect.right, add_valve_rect.bottom + 2 * height);

// shadow hut control
a.v = mode_rect.bottom + 5;
SetRect(&shadow_rect, a.h, a.v, a.h + width, a.v + height);
SetRect(&shadow_rect_contrect, shadow_rec.right - 15, shadow_rec.top + 5, shadow_rec.right - 5, shadow_rec.bottom - 5);

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

// simulated speed
a.v = pattern_rec.bottom + 20;
width = 15;
height = 15;
for(i = 0; i < 10; i++)
    SetRect(&pattern_rect_contrect, 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(&image_map_contrect, a.h, a.v, a.h + 50, 420);
SetRect(&image_map_leftrec, a.h, a.v, a.h + width, a.v + height);
SetRect(&image_map_rightrect, a.h, a.v + height + 10, a.h + width, a.v + 2 * height + 10);
SetRect(&image_map_toprect, a.h, a.v + 4 * height, a.h + width, a.v + 5 * height);
SetRect(&image_map_bottomrect, a.h, a.v + 5 * height + 10, a.h + width, a.v + 6 * height + 10);
SetRect(&image_map_leftrect, a.h, a.v + 7 * height, a.h + width, a.v + 8 * height);
SetRect(&image_map_top_rect, a.h, a.v + 8 * height + 10, a.h + width, a.v + 9 * height + 10);
SetRect(&image_map_rect, a.h, a.v + 9 * height + 20, a.h + width, a.v + 10 * height + 20);
SetRect(&image_map_rect_left, a.h, a.v + 10 * height + 30, a.h + width, a.v + 11 * height + 30);

// run mode
my_fill_rect(&mode_rec, INSET_BG, 1);
my_draw_rect(mode_rec, BLACK);
do_text_rec(mode_rec, "AUTO", 0);
my_fill_rect(&mode_control_rec, BLACK);
my_draw_rect(&mode_control_rec, WHITE, 1);

// shadow rect
my_fill_rect(&shadow_rec, INSET_BG, 1);
my_draw_rect(shadow_rec, BLACK);
do_text_rec(shadow_rec, "SHADOWS", 0);
my_draw_rect(shadow_control_rec, BLACK);
my_draw_rect(shadow_control_rec, WHITE, 1);

// expand pattern
my_fill_rect(&pattern_rec, INSET_BG, 1);
my_draw_rect(pattern_rec, BLACK);
do_text_rec(pattern_rec, "PATTERN", 0);
my_draw_rect(pattern_control_rec, BLACK);
my_fill_rect(&pattern_control_rec, WHITE, 1);

// simulated speed
for(i = 0; i < 10; i++)
    }
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_rect(&valve_auto_rect[v], BLACK);
      break;
    case MANUAL_ON:
      my_fill_rect(&valve_on_rect[v], DKGRAY, 1);
      my_draw_rect(&valve_on_rect[v], BLACK);
      break;
    case MANUAL_OFF:
      my_fill_rect(&valve_off_rect[v], DKGRAY, 1);
      my_draw_rect(&valve_off_rect[v], BLACK);
      break;
  }
  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 l;
  char strvig[50] = "0"
  Rect info_rect;
  short width = 100;
  short height = 20;
  Point a, b;
  a.y = 100;
  a.x = 150;
  b.x = a.x + band_count * height;
  HideCursor();
  // frame
  SetRect(&fr, band_rect[0].left, band_rect[0].top, band_rect[0].right, band_rect[0].bottom);
  my_draw_frame(&fr, fw);
  SetRect(&fr, near_side_rect.left, near_side_rect.top, near_side_rect.right, near_side_rect.bottom);
  my_draw_frame(&fr, fw);
}
// draw finished rectangle
my_fill_rect(&next_map_set_rect, WHITE, 1);
my_draw_frame(&next_map_set_rect, tw);
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, tw);
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, tw);

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

my_fill_rect(&image_map_hsepixel_rect, WHITE, 1);
my_draw_frame(&image_map_hsepixel_rect, tw);

my_fill_rect(&image_map_vsepixel_rect, WHITE, 1);
my_draw_frame(&image_map_vsepixel_rect, tw);

my_fill_rect(&image_map_hmenu_rect, WHITE, 1);
my_draw_frame(&image_map_hmenu_rect, tw);

draw_up_button(more_hmenu_rect, INSET_BG);
draw_down_button(less_hmenu_rect, INSET_BG);

my_fill_rect(&image_map_vmenu_rect, WHITE, 1);
my_draw_frame(&image_map_vmenu_rect, tw);

draw_up_button(more_vmenu_rect, INSET_BG);
draw_down_button(less_vmenu_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, 13);

// draw control buttons(Rect rect, 10, 10);

// void draw_vd_vbi2_lines(Bool *status)
// Rect fr;
// char tw = 2;

HideCursor();

if (status) {

// draw finish rectangle
my_fill_rect(&vd_adjust_rect, INSET_BG, 1);
my_draw_frame(&vd_adjust_rect, tw);

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

draw_up_button(vd_more_contrast_rect, INSET_BG);
draw_down_button(vd_less_contrast_rect, INSET_BG);

my_fill_rect(&vd_brightness_rect, INSET_BG, 1);
my_draw_frame(&vd_brightness_rect, tw);
draw_up_button(vd_more_brightness_rect, INSET_BG);
draw_down_button(vd_less_brightness_rect, INSET_BG);

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

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

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

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, "$\text{Video}$", 0);

fr.left = vd_source_rect.left;
fr.top = vd_source_rect.bottom;
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, "$\text{Composite}$", 0);

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

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, "$\text{Interface}$", 0);

fr.left = vd_mode_rect.left;
fr.top = vd_mode_rect.bottom;
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, "$\text{Even/Odd}$", 0);

my_fill_rect(&vd_lut_rect, INSET_BG, 1);
my_draw_frame(&vd_lut_rect, tw);

fr.left = vd_lut_rect.left;
fr.top = vd_lut_rect.top;
fr.right = vd_lut_rect.right;
fr.bottom = vd_lut_rect.bottom;
do_text_rect(fr, "$\text{Apply LUT}$", 0);
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;
else
    h_center = rect.left + 2, v_center + 4;
drawstring(print_string);
}

void draw_up_button(Rect rect, unsigned long color)
{
    short x1, y1, x2, y2;
    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.top + rect.top) / 4;
    x1 = x2 = mid;
    HideCursor();
    my_fill_rect(&rect, color, 1);
    my_draw_rect(rect, BLACK);
    for (y = ystart; y < yend; y++)
    {
        for (x = x1; x < x2; x++)
            my_draw_point(x, y, BLACK);
        x1++; x2++; 
    }
    my_draw_frame_int(&rect, 2);
    my_draw_rect(rect, BLACK);
    ShowCursor();
}

void draw_down_button(Rect rect, unsigned long color)
{
    short x1, y1, x2, y2;
    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.top + 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_int(&rect, 2);
    my_draw_rect(rect, BLACK);
    ShowCursor();
}

my_draw_frame_int(&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, DKGREY);
    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 *cp;
    static short last_status=0, last_status2=0, last_status3=0;
    short status;
    unsigned char din_c;
    char warning[4];
    // do overspeed warning at operator console
    read_din(C, &din_c);
    write_din(C, din_c | OVER_SPEED);

    /*
     * select location and color for type of overspeed warning on monitor
     * switch(warning_type)
     */
    case(0):
        last_status = (last_status) ? 0 : 1;
        *status = last_status;
        warning_rect = &overspeed_rect0;
        color = 0x00f00000;
        skip = 1;
        sprintf(warning, "V");
break;
case (1):
  last_status1 = (last_status1) ? 1:
  *status = last_status1;
  warning_rect = &overspeed_rect1;
  color = 0x00000000;
  skip = 1;
  sprintf(warning, "1");
  break;
case (2):
  last_status2 = (last_status2) ? 1:
  *status = last_status2;
  warning_rect = &overspeed_rect2;
  color = 0x0000000f;
  skip = 1;
  sprintf(warning, "P");
  break;
case (3):
  last_status3 = (last_status3) ? 1:
  *status = last_status3;
  warning_rect = &overspeed_rect3;
  color = 0x0000ff00;
  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 Tauer
// 1995
// This file contains the functions used for controlling the TAOS system using computer
// input.
// ---------

EventRecord gEvent;
short gwindowCode;
Boolean guserDone;
CursorHandle gbCursor;
WindowPtr gwindow;

extern WindowPtr subWindow, mainWindow;
#ifdef UNIX
  extern unsigned long firstTick, last_count;
#endif
extern unsigned long overflow_record;

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_list(unsigned long base);
extern void do_nice_list(void);
extern void do_image_adjust(void);
extern void do_roll_adjust(void);
extern void do_valve_adjust(void);
extern void do_image_mapping(void);
extern void do_image_adjust(short rd_number, short *br, short *cr, 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 & 0x0f) != last_din_c)
    {
      // update digital input record
      last_din_c = din_c & 0x0f;
      // 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, "OE", 0);
    if (!WaitNextEvent(everyEvent, &gEvent, LONG_NAP, NO_CURSOR);
      if (gEvent->what == DoEvent();
        while(status.run);
    }
    // ---------
    // check for user input

Copyright 2011, AHMCT Research Center, UC Davis
Copyright © 1995 by University of California Davis
void DoEvent()
{
    // call appropriate routines
    //
    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:
            btnSet(gEvent.where);
            break;
        }
        break;
    // keyboard input
    case keyDown:
        if((gEvent.modifiers & cmdKey) == 0)
            guserDone = Do_Command(MenuKey((char *)&gEvent.message & charCodeMask));
        else
            break;
    
    case 'c':
        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.expanded);
    
    // computer controls
    for(i = 0; i < 10; i++)
        if(mouse_in(sixSpeedRect[i], where))
            simSpeed = (unsigned long)(i + 1);
    do_simulated_speed_control();
    
    // 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();
    }
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 VALUE_MENU:
break;
case RUN_MODE_MENU:
switch(theItem)
|
case PIXEL_DISPLAY:
status.show_pixels = (status.show_pixels + 1) % 2;
break;
case DATA_DISPLAY:
status.show_data = (status.show_data + 1) % 2;
if(status.show_data == 1)
draw_data_area(CN);
else
draw_data_area(OFF);
break;
|
}
HiLiteMenu(0);
return quitApp;
// .................................................................
//  

Copyright 2011, AHMCT Research Center, UC Davis

Copyright © 1995 by University of California Davis