THE SOFTWARE DOCUMENTATION
OF
POSITION REFERENCE SYSTEM
A User's Manual

Ver. 1.0

By

Sunil Fotedar

Contract NAS 9-17900
Job Order 16-403

ABSTRACT

All the routines used in developing software for Position Reference System are discussed in this documentation.

ACKNOWLEDGEMENTS

Thanks are due to Oxford and Associates, Inc. of Houston; in particular Mr. Matthew W. Prucka; for developing software for the system.

CONTENTS

CONTENTS (Contd.)

CONTENTS (Contd.)

INTRODUCTION

This report forms the second part of the three-part documentation of the Position Reference System (PRS). Here we shall present all the routines used in developing software for the PRS. This report will be split up into three categories, namely :

Section A. Software for Position Reference System
Section B. Position Reference System Source Code Reference Manual
Section C. Library Reference Manual.

In section A, the software developed for PRS will be discussed. Section B explains all the routines used for PRS software. Section C gives an account of all the library routines used in the development of the software.

SECTION A

SOFTWARE FOR POSITION REFERENCE SYSTEM

2.1 HOW TO GET STARTED

The high-density diskette, provided with *THE SOURCE CODE FOR PRS* (Ver. 1.0), contains all the software for running PRS. Put this diskette in the drive A of the main computer (ref.[1]). At the prompt, type:

RESTORE

It is a batch file created to copy all the files from the diskette to drive C: (fixed disk drive of the main computer). Similarly, if a backup is desired, put a new formatted high-density diskette into drive A: and at the prompt, type:

BACK

In order to run the *PRS* program, change the directory of the main computer to the *\prs* directory by typing the following command:

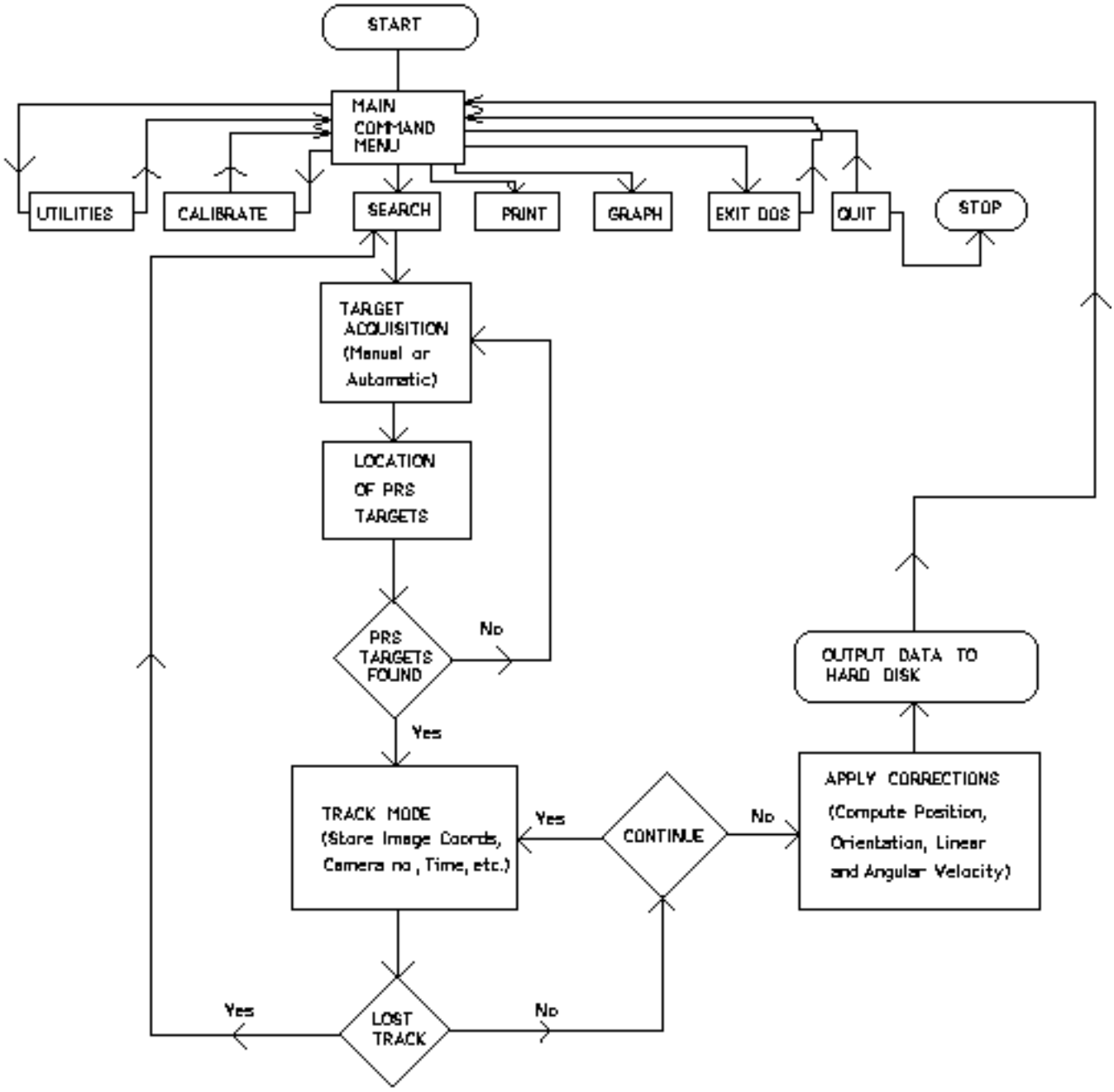
CD PRS

At the prompt type:

PRS

and hit <RETURN> key, which will begin execution of the program. The first window we see is the *Main Command Menu* (MCM) window (Fig. 2.1). This window will allow user to select one of several main functions. The purpose of the PRS is to locate three bright spots in an image of one of the eleven cameras installed in the llaboratory, and then track them.

Fig. 2.1: Flow Chart for PRS Software Development.



It is done to find their *state vector* (position, orientation, linear and angular velocities and accelerations). The bright spots in an image represent the *PRS targets* on a robot. The robot is remotely controlled by a computer. The PRS targets are kept in a right-angled triangular configuration and a dark background is provided. In order to search for the target and then track, the user should select the *Search* command from the MCM window. This is done by either pressing the *S* key or by highlighting the word by means of cursor keys and pressing the <RETURN> key. Once in the search mode, the program will automatically search for the bright spots in each camera's field of view (FOV). Once the targets are found, the program will begin to track the target. The user may abort this operation with the <ESC> key or <F10> key. When the user leaves the *Search* command, the program will return to the MCM window. Here, the user may select other commands, namely; *Utilities*, *Calibrate*, *Print*, *Graph*, *Exit DOS*, or *Quit* commands.

If some modifications are made to the program, the user may recompile all source modules which have changed by using the Microsoft's *MAKE* utility. This is done by typing:

MAKE PRS.MAK

2.2 PROGRAM STRUCTURE

The *PRS* program is structured in hierarchical set of command windows. When the program starts executing, the user is placed in the MCM window. This window allows the user to select any of the major commands in the program. Selecting a command may be done by either highlighting the command by using the up and down cursor keys and then

hitting <RETURN> key, or by hitting the key that corresponds to the highlighted letter of the command (in MAGENTA color in VGA mode). The program consists of seven commands as follows:

* Utilities
* Calibrate
* Search
* Print
* Graph
* Exit DOS
* Quit

A discussion of each command follows:

2.2.1 Utilities

This command gives the user access to many useful functions. Some of these functions take place on the display monitor (color), both in text and graphics modes, and the camera display monitor (monochrome). Here the user is able to change the following global parameters of the program:

Current Camera Field: The parameter contains the number of the camera with which all operations will be performed. The user may select any camera simply by entering a camera value in this field. The default camera number is 6.

Threshold Level: The user may also set the current threshold level which is used when PRS targets are searched. The default value is 120.

Maximum Background: The user may set this parameter in order to specify background. The default value is 0.

Bias: The parameter indicates the bias level which is set before a snap shot of the current camera is taken. The default value is 0.

Image File Name: This field allows the user to specify a file name, usually *TEST.IMG* , which is used when image files are written to/read from the disk.

Recorder File Name: This field allows the user to specify a file name, usually *D:RECORDER.DAT* , which is used to record the history of a given target track in terms of the 2-D image co-ordinates of the PRS target(s). The D: drive is used as a *virtual* disk in order to speed up the computer while the PRS target is being tracked.

The user can choose various commands from this window, displayed at the bottom of the EGA screen, by simply hitting a function key. The functions of these keys are described as follows:

Snap Shot (<F1>):
By pressing <F1>, the user will take a *snap shot* of the scene using the current camera which is displayed on the screen. When a snap shot is taken, the video image will appear on the camera display only and the image is not placed in the computer's HUGE image buffer. Therefore, before any analysis is desired on a current snap shot, the image must first be

placed in the buffer by the *MVP->VGA* function.

Continuous (<F2>):

This function will place the Matrox MVP-AT/NP frame grabber in *continuous frame grabbing* mode. The user may exit this mode by simply pressing any key.

Initialize (<F3>):

This function will *initialize* the frame grabber and it only needs to be done if the display on the camera does not look desirable. Sometimes this function must be done more than once.

MVP->VGA (<F4>):

This function will place the image currently in the frame grabber into the HUGE image buffer and then display that buffer in VGA graphics mode. See *Buff.->VGA*.

Buff.->VGA (<F5>):

This function will place the computer terminal into VGA graphics mode and place an image of the current HUGE image buffer on the screen. The image is placed on the screen using a 16 pseudo-color scale. Also, the image has been compressed by a factor of 4 (2 in both x and y directions). Once the image is on the screen of the display monitor, the user will notice a graphics cursor which is controlled by a mouse. By moving the mouse, the cursor can be pointed to any pixel in the image. The computer will display the actual four image points averaged to one point. The computer will also display the *boundary values* for each of the four pixels. A *pass* or *fail* will then be displayed for each pixel depending on the results of the *boundary test* for that pixel (ref. [1]).

The user may perform a search on the image by pressing <F1>. If the computer is able to locate the target, it will display the corresponding target points by drawing a circle around each target which is found on the screen. To leave the VGA graphics display, the user should press <F10> or <ESC>.

MVP->File (<F6>):
This function will read the image file name which is displayed on the screen and place the image currently on the video display to that file.

File->EGA (<F7>):
This function will place an image which is stored in the image file in the HUGE image buffer and then display that buffer in VGA graphics mode. See *Buff.->VGA*.

Search (<F9>):
This function will perform a *search* in the HUGE image buffer for the PRS targets. If the computer is able to locate the target, it will display the corresponding targets by drawing a circle around each on the camera display monitor.

Return (<F10> or <ESC>):
The user will be able to leave this window using either this command or by pressing <ESC>.

2.2.2 Calibrate

This command allows user to specify a file, named *XY_UV.DAT*, which contains the (x, y) and corresponding (u, v, z) co-ordinates of the *optical targets* (calibration points) along

with the corresponding camera for the (x, y) co-ordinate of the calibration point. The data is stored by the user in the following format:

$x_{1,j}$	$y_{1,j}$	$u_{1,j}$	$v_{1,j}$	$z_{1,j}$	c_j	$prs\_u_{1,j}$	$prs\_v_{1,j}$
$x_{2,j}$	$y_{2,j}$	$u_{2,j}$	$v_{2,j}$	$z_{2,j}$	c_j	$prs\_u_{2,j}$	$prs\_v_{2,j}$
$x_{3,j}$	$y_{3,j}$	$u_{3,j}$	$v_{3,j}$	$z_{3,j}$	c_j	$prs\_u_{3,j}$	$prs\_v_{3,j}$
*	*	*	*	*	*	*	*
*	*	*	*	*	*	*	*

where ($x_{n,j}$, $y_{n,j}$) and ($u_{n,j}$, $v_{n,j}$, $z_{n,j}$) are the image and floor co-ordinates of the n^{th} (n > 15) optical target respectively in j^{th} (j=1,2,...,11) camera's FOV. ($prs\_u_{1,j}$, $prs\_v_{1,j}$) are the designated PRS co-ordinates of the optical targets, which are not used for calibration purposes and are stored in the file only to keep the user informed about which target is being referred to. This file may contain as many calibration points as is desired. The floor co-ordinates of the targets are measured by means of high-precision electronic theodolites.

The calibration process yields a set of co-efficients ($A_{i,j}$, $B_{i,j}$) for each camera (i=0,1,2,...,15; j=1,2,3,...,11). These co-efficients are stored in a separate file, called *CALIBRAT.DAT* , and are used later when the image co-ordinates of the tracked PRS target(s) are converted into height-corrected floor co-odinates to give the *position* of the PRS target.

The user can choose various commands from this window by simply hitting a function key. The functions of these keys are described as follows:

Calibrate (<F1>):
By pressing <F1>, the data from *XY_UV.DAT* will be used to calibrate each camera and the result put into *CALIBRAT.DAT*.

View (<F2>):
By pressing <F2>, the user can view either of the three files - *XY_UV.DAT*, *CALIBRAT.DAT* or *D:RECORDER.DAT* on the screen. Any of these files can be selected by hitting the *SPACEBAR* of the keyboard of the main computer. The DOS command *TYPE filename|MORE* has been used for this purpose.

Return (<F10> or <ESC>):
The user may return to the MCM window by pressing <F10> or <ESC>.

2.2.3 Search

When the user selects this command, the PRS program will begin by searching the current camera's output (which can be changed in the *Utilities* command) for the target. The user has the option to track only one target (to get information about position, linear velocity and linear acceleration) or two targets (to determine orientation, angular velocity and angular acceleration in addition to position, linear velocity and linear acceleration). The targets are identified as target #1, target #2, and target #3 depending on which side (of the triangle they form) they are opposite to. If the target(s) is not found in the current camera, the program will change cameras automatically and continue searching for the target(s) until it is found, or the user aborts by pressing either <ESC> or <F10>.

Various commands can be chosen from this window by hitting some function keys. The

function of these keys is described as follows:

File ON/OFF (<F2>):

The 2-D image co-ordinates of the target(s) and the history of the camera selection is written to *D:RECORDER.DAT* . This function may be turned *off* or *on* simply by pressing <F2>.

Manual (<F3>):

By pressing <F3> while the target is being searched, the user may place the cursor by the mouse over a given camera field in the *camera layout* being displayed on the screen. The 'camera layout' is a representative diagram of the FOV of each camera. Once the user has placed the mouse cursor in the FOV of the desired camera, the user may select that camera to search for the target by hitting either of the mouse buttons.

Return (<F10> or <ESC>):

The user may leave window by hitting either <ESC> or <F10>. This will return the user to the MCM window. While exiting this window, the user will have the option to convert the tracked 2-D image co-ordinates into floor co-ordinates and stored in various other data files. Other computations are also performed to find position, orientation, linear and angular velocities and accelerations (or state vector). The data files created, with their contents (in addition to camera number and time), are listed in the table 2.1.

2.2.4 Print

This function will allow the user to print any one of the data file by first selecting the data file with the *SPACEBAR* of the keyboard and then pressing the <F1> key. The user can choose various commands from this window by simply hitting a function key. The functions

of these keys are described as follows:

Print (<F1>):

By pressing <F1>, the user can print either of the the files - *TARGET1.DAT,*

Table 2.1: Various Data Files and their contents

Data File	Contents
D:RECORDER.DAT	2-D image co-ordinates of target(s)
TARGET1.DAT	Position of target #1
TARGET2.DAT	Position of target #2 (if tracked)
VEL_ACC.DAT	Linear velocity and acceleration of target #1
ANGLE.DAT	Orientation, angular velocityand acceleration of the targets (only if target #2 is tracked)

TARGET2.DAT, VEL_ACC.DAT, ANGLE.DAT, or *D:RECORDER.DAT* . The NORTON utility *LIST filename* has been used for this purpose.

View (<F2>):

By pressing <F2>, the user can view the three files - *XY_UV.DAT*, *CALIBRAT.DAT* or *D:RECORDER.DAT* on the screen. The DOS command *TYPE filename|MORE* has been used for this purpose.

Stop (<F9>):
Any data sent to the printer for printing can be stopped at any moment by pressing <F9>.
It is particularly useful when the printer is printing out garbage or anything that the user does not like.

Return (<F10> or <ESC>):
The user may return to the MCM window by pressing <F10> or <ESC>.

2.2.5 Graph

This function will allow the user to plot the data in various data files on a plotter, which is connected to the main computer through the secondary communications port. The various options available to the user are:

Plot (x,y) (<F1>):
By pressing <F1>, the user plots the 2-D image co-ordinates (x,y) of the PRS target(s) with respect to time (in terms of *elapsed seconds*).

Target#1 (<F2>):
The *height-projected* floor co-ordinates (u_T, v_T) of target #1 (position) are plotted as a function of time by pressing <F2>.

Target#2 (<F3>):
The height-corrected floor co-ordinates of target #2 (if tracked) are plotted as a function of time by pressing <F3>.

Lin. Vel. (<F4>):
By pressing <F4>, the *linear velocity* of target #1 is plotted against time.

Lin. Acc. (<F5>):
The *linear acceleration* of target #1 is plotted against time by pressing <F5>.

Angle (<F6>):
The *orientation* of the robot can be plotted as a function of time by pressing <F6>.

Ang. Vel. (<F7>):
The *angular velocity* of the robot is plotted against time if <F7> is pressed.

Ang. Acc. (<F8>):
The *angular acceleration* of the robot is plotted against time if <F8> is pressed.

Return (<F10> or <ESC>):
The user may return to the MCM window by either pressing <F10> or <ESC>.

2.2.6 Exit DOS

This command allows the user to create a DOS session which can be used to access the operating system commands. In order to return to the program which remained in

memory, just type:

EXIT

and hit the <RETURN> key.

Quit

The user may terminate and leave the program by using this command.

2.3 FILE DESCRIPTION

This section will describe the files which contain various routines to develop software for PRS.

2.3.1 Source Code

All source files are in the directory */prs* .

cal.c
This file contains the calibration routine of the cameras.

convert.c
This file contains all the routines required for conversion of 2-D image locations of the target(s) into height-corrected position co-ordinates, and determination of

state vector of the target(s).

display.c

Contains the code required to place a camera image to the VGA screen.

frame.c

This routine draws the border around the menus and displays the title information.

hp_line.c

This contains the plotter routines.

iocom1.c

Serial port 1 communication routine used to establish link with the video switcher.

iocom2.c

Serial port 2 communication routine used to establish link with the plotter.

matrix.c

Several matrix operations including matrix transpose, matrix multiply, and matrix inverse.

matrox.c

Contains an initialization routine used on the frame grabber. This file also contains dummy routines when running the program from computer which does not have a frame grabber.

mouse_f.c

Routines used in setting up and controlling the mouse.

plot1.c , plot2.c, plot3.c, plot4.c

Contain routines for plotting various components of state vector of the targets are in these files.

print.c

This routine will allow the user to print out data files.

prs.c

The entry point of the program. The main menu is located in this file.

search.c

Routines used to search for PRS targets in an image.

swcam.c

Routines used by the video switcher to select cameras.

track.c

This files contains the routines used to track the PRS target(s).

util.c

Various routines used in testing the cameras and the search routine.

2.3.2 Include Files

The following `include` files are in the directory *\prs\include* because they are directly related to the main program.

mvp.h

All of the definition statements associated with the frame grabber are included in this file.

prs.def

This file contains all the # `define` statements for all the source files.

prs.h

This file contains the declarations of all `global` variables. This file is only included in the main source routine.

prs.ext

This file contains `extern` declarations of all the variables which are declared in *prs.h*. This file is included in any routine which used any `global` variables.

The rest of the `include` files are in the directory *\prs\library\include* because they are associated with the library routines.

box.def

This file contains the ASCII values for the box drawing characters.

color.def

The values for *text* and *graphic* colors are here.

command.ext

The `extern` declarations for the command structure which is used in the command-handler routines.

command.h

The `global` declarations of the preceding file.

menu.ext

The `extern` declarations for the menu structure which is used in the menu-handler routines.

menu.h

The `global` declarations of the preceding file.

mouse.def

#`define` statements for the mouse -handler routines.

scancode.def

Keyboard scancode #`define` values.

screen.ext

The `extern` declarations of the screen structure which is used in the screen- handler routines.

screen.h

The `global` declarations of the preceding file.

2.3.3 Library Files

mvplm.lib

The frame grabber library in the directory *\prs*.

mplm.lib

The frame grabber library in the directory *\prs*.

mouse.lib

This is the *MICROSOFT C* library from the *Microsoft's Mouse Development Kit (Version 1.01)*. The source file *mouse_f.c* needs this library. This library can be found in the default library directory which is defined in the *AUTOEXEC.BAT*.

library.lib

This is a library of a set of miscellaneous C routines. The library can be found in *\prs\library\lib*.

2.3.4 Make Files

prs.mak

This file contains the *MAKE* utility of the *Microsoft's C Compiler (ver. 5.0)* which is used to *compile/link* all of the source code.

prs.arf

This file contains the *link* parameters for the program.

SECTION B

PRS SOURCE CODE REFERENCE MANUAL

BACKGROUND_CHECK

DESCRIPTION:

This function will calculate the background threshold value. *Offset* is the number of (bytes in a row * size of the target) which will be used to find where the vertical pixels will start for the background. background_value[0] is the left horizontal; [1] is the right horizontal; [2] is up vertical; [3] is down vertical.

SYNTAX:

```
int background_check( current_pixel, offset )

unsigned char huge *current_pixel;
int offset;
```

CALIBRATE

DESCRIPTION:

This function will allow the user to calibrate the CCD cameras.

SYNTAX:

```
int calibrate()
```

CALIBRATE_CAMERAS

DESCRIPTION:

This function will calculate the *calibration co-efficient matrix* for each CCD camera and store that matrix in the file which is specified by the user (CALIBRAT.DAT).

SYNTAX:

calibrate_cameras()

CALIBRATE_MENU

DESCRIPTION:

This function will set the menu structure for the calibrate command window.

SYNTAX:

calibrate_menu()

CONVERT_TRACK_FILE

DESCRIPTION:

This function will convert the tracked 2-D image co-ordinates into floor co-ordinates by using the calibration co-efficient matrix for each camera. This gives position of one of the targets and its linear velocity and acceleration are computed (if *track_two_targets* is NO). If two targets are tracked (track_two_targets = YES), the orientation and angular velocity and acceleration of the targets are also computed.

SYNTAX:

```
convert_track_file(track_two_targets)

int     track_two_targets;
```

DISK_TO_MVP

DESCRIPTION:

This function will write a 512x512 byte array to the frame grabber board and from image_file_name. SUCCESSFUL (0) will be returned, if appropriate.

SYNTAX:

```
int disk_to_mvp()
```

DISPLAY_IMAGE

DESCRIPTION:

This function will display an image on the VGA graphics screen. The image is either FROM_FILE (0) or FROM_MVP (1) or FROM_BUFFER (2).

SYNTAX:

```
int display_image( origin )
int origin;
```

FAST_AT

DESCRIPTION:

This function will speed up the PC's Limited 286 to 12 Mz. This is not needed if Compaq's 386/20 is used.

SYNTAX:

fast_at()

FRAME

DESCRIPTION:

This function will display the menu-style frame.

SYNTAX:

frame()

GET_STOP_WATCH

DESCRIPTION:

This function will get the *current* and *elapsed* stop-watch time.

SYNTAX:

get_stop_watch()

HP_DRAW

DESCRIPTION:

This void function will plot a line on an HP plotter.

SYNTAX:

```
void  hp_draw(x1, y1)

int   x1, y1;
```

HP_MOVE

DESCRIPTION:

This void function will move the HP plotter pen.

SYNTAX:

```
void  hp_move(x1, y1)

int  x1, y1;
```

HP_PEN_DOWN

DESCRIPTION:

This void function will set the HP plotter pen down.

SYNTAX:

```
void  hp_pen_down( )
```

HP_PEN_UP

DESCRIPTION:

This void function will lift the HP plotter pen up.

SYNTAX:

```
void  hp_pen_up( )
```

INIT_COM1

DESCRIPTION:

This function will initialize serial port 1.

SYNTAX:

```
init_com1( )
```

INIT_COM2

DESCRIPTION:

This function will initialize serial port 2.

SYNTAX:

```
init_com2( )
```

INITIALIZE_MVP

DESCRIPTION:

This function will initalize the Matrox's MVP AT/NP frame grabber board.

SYNTAX:

int initialize_mvp()

MANUAL_SELECT_CAMERA

DESCRIPTION:

This function will allow the user to pick a new camera with the mouse. The routine will return the new_camera selected and the approximate coordinates in the camera image.

SYNTAX:

```
manual_select_camera( new_camera, x, y )

int *new_camera;
int *x, *y;
```

MATRIX_INVERSE

DESCRIPTION:

This function will calculate the inverse of a given matrix *mat[]* and return the result in *inv[]*. The matrix is a square matrix with dimension *dim*.

SYNTAX:

```
double matrix_inverse( mat, dim, inv )

double  mat[] ;
double  inv[] ;
int     dim ;
```

MATRIX_MULTIPLE

DESCRIPTION:

This function will calculate the matrix multiplication of matrix *a* with matrix *b* and place the result in matrix *c*.

SYNTAX:

```
void matrix_multiple( a, b, c, row_in_a, col_in_a, col_in_b )

double *a;
double *b;
double *c;
int row_in_a, col_in_a, col_in_b ;
```

SEMANTICS:

This routine is to do matrix multiply of a * b = c. The inputs should be declared in main as:

```
double a[ "row_in_a" ][  "col_in_a" ] ; double b[ "col_in_a"]
        [ "col_in_b" ] ; double c[ "row_in_a" ][ "col_in_b" ] ;
```

The call would then be:

```
matrix_multiple( a, b, c,  "row_in_a",  "col_in_a", "col_in_b" ) ;
```

EXAMPLE:

```
double a[2][3], b[3][3], c[2][3] ;

matrix_multiple( a, b, c, 2, 3, 3 ) ;
```

Note Well: If the variables row_in_a, col_in_a, and col_in_b do not exactly match those dimensions of their arrays when declared, then these routines will not work!

MATRIX_TRANSPOSE

DESCRIPTION:

This function will calculate the matrix transpose of matrix a and place the result in matrix b.

SYNTAX:

```
void matrix_transpose( a, b, row_in_a, col_in_a )

double *a, *b ;
int row_in_a, col_in_a ;
```

SEMANTICS:

This routine transposes the matrix a and places the result in matrix b.

Note Well: The row_in_a and col_in_a variables must be equal to the row and col dimensions of the matrix a, and to col and row dimensions of the matrix b.

EXAMPLE:

```
double a[2][3], b[3][2] ;

matrix_transpose( a, b, 2, 3 ) ;
```

MOUSE_GET_BUTTON_PRESS

DESCRIPTION:

This function will return the current status of the buttons in status with bit 0 representing the left button and bit 1 the right button. If the bit is a 1 then the button is down; 0 the button is up. The user will specify which button the information is requested in the variable button. The routine will return the number of times the button was pressed since the last time this routine was called. Also, the horizontal and vertical position of the cursor at the last time the button was pressed is returned.

SYNTAX:

```
mouse_get_button_press( button, status, number_of_presses,
                                        horz_pos, vert_pos)

int button;
int *status;
int *number_of_presses;
int *horz_pos, *vert_pos;
```

MOUSE_GET_STATUS

DESCRIPTION:

This function will get the mouse button status and mouse position.

SYNTAX:

```
mouse_get_status( status, horz_position, vert_position )

int *status;
int *horz_position;
int *vert_position;
```

MOUSE_GRAPHIC_CURSOR

DESCRIPTION:

This function will display a graphics cursor.

SYNTAX:

mouse_graphics_cursor()

MOUSE_HIDE_CURSOR

DESCRIPTION:

This function will turn off the mouse cursor.

SYNTAX:

```
mouse_hide_cursor()
```

MOUSE_RESET

DESCRIPTION:

This function will reset the mouse. This function will return MOUSE_NOT_INSTALLED (-1) if appropriate.

SYNTAX:

```
int mouse_reset()
```

MOUSE_SET_CURSOR

DESCRIPTION:

This function will set the mouse cursor position.

SYNTAX:

```
mouse_set_cursor( horz_position, vert_position )

int horz_position;
int vert_position;
```

MOUSE_SET_HORZ_LIMIT

DESCRIPTION:

This function will set the minimum and maximum horizontal cursor position.

SYNTAX:

```
mouse_set_horz_limit( minimum_position, maximum_position )

int minimum_position;
int maximum_position;
```

MOUSE_SET_MICKEY

DESCRIPTION:

This function will set the mouse mickey to pixel ratio.

SYNTAX:

```
mouse_set_mickey( horz_mickey, vert_mickey )

int horz_mickey;
int vert_mickey;
```

MOUSE_SET_TEXT_CURSOR

DESCRIPTION:

This function will set the text cursor. If cursor_select equals 0 then the software text cursor will be selected, else if the cursor_select equals 1 the hardware text cursor is selected. The mask variable is defined as:

bits(15-blinking, 12-14 background color, 11 intensity,
8-10 foreground color, 0-7 character).

The screen_mask is ANDed with the current char. and cursor_mask is XORed with the result. If hardware cursor is selected, the screen_mask is the scan line start and the cursor_mask is the scan line stop.

SYNTAX:

```
mouse_set_text_cursor( cursor_select, screen_mask, cursor_mask )

int cursor_select;
int screen_mask, cursor_mask;
```

MOUSE_SET_VERT_LIMIT

DESCRIPTION:

This function will set the minimum and maximum vertical cursor position.

SYNTAX:

```
mouse_set_vert_limit( minimum_position, maximum_position )

int minimum_position;
int maximum_position;
```

MOUSE_SHOW_CURSOR

DESCRIPTION:

This function will turn on the mouse cursor which will automatically move with the mouse movement.

SYNTAX:

mouse_show_cursor()

MVP_TO_DISK

DESCRIPTION:

This function will read a 512 x 512 byte array from the MVP board and put it into image_file_name. SUCCESSFUL (0) will be returned if appropriate.

SYNTAX:

int mvp_to_disk()

PLOT_FIND_ACC_U_MAX_MIN

DESCRIPTION:

This function will find minimum and maximum of 'u' component of linear acceleration in *filename* .

SYNTAX:

```
plot_find_acc_u_max_min(plot_acc_u_min,plot_acc_u_max, filename)

char    *filename;
int     *plot_acc_u_min;
int     *plot_acc_u_max;
```

PLOT_FIND_ACC_V_MAX_MIN

DESCRIPTION:

This function will find minimum and maximum of 'v' component of linear acceleration in *filename* .

SYNTAX:

```
plot_find_acc_v_max_min(plot_acc_v_min,plot_acc_v_max, filename)

char    *filename;
int     *plot_acc_v_min;
int     *plot_acc_v_max;
```

PLOT_FIND_ANG_ACC_MAX_MIN

DESCRIPTION:

This function will find minimum and maximum of angular acceleration in *filename* .

SYNTAX:

```
plot_find_ang_acc_max_min(plot_ang_acc_min,plot_ang_acc_max,
                                        filename)

char    *filename;
int     *plot_ang_acc_min;
int     *plot_ang_acc_max;
```

PLOT_FIND_ANG_VEL_MAX_MIN

DESCRIPTION:

This function will find minimum and maximum of angular velocity in *filename* .

SYNTAX:

```
plot_find_ang_vel_max_min(plot_ang_vel_min,plot_ang_vel_max,
                                        filename)

char    *filename;
int     *plot_ang_vel_min;
int     *plot_ang_vel_max;
```

PLOT_FIND_MAX_TIME

DESCRIPTION:

This function will return the maximum elapsed time in *filename*.

SYNTAX:

```
plot_find_max_time( filename )

char *filename;
```

PLOT_FIND_MAX_TIME_ANGLE

DESCRIPTION:

This function will return the maximum elapsed time in *filename*.

SYNTAX:

```
plot_find_max_time_angle( filename  )

char      *filename;
```

PLOT_FIND_MAX_TIME_VEL_UV

DESCRIPTION:

This function will return the maximum elapsed time in *filename*.

SYNTAX:

```
plot_find_max_time_vel_uv( filename )

char *filename;
```

PLOT_FIND_V_MAX_MIN

DESCRIPTION:

This function will find minimum and maximum of 'v' co-ordinates in *filename* .

SYNTAX:

```
plot_find_v_max_min( plot_v_min, plot_v_max, filename)

char    *filename;
int     *plot_v_min;
int     *plot_v_max;
```

PLOT_FIND_VEL_U_MAX_MIN

DESCRIPTION:

This function will find minimum and maximum of 'u' component of linear velocity in *filename* .

SYNTAX:

```
plot_find_vel_u_max_min(plot_vel_u_min,plot_vel_u_max, filename)

char    *filename;
int     *plot_vel_u_min;
int     *plot_vel_u_max;
```

PLOT_FIND_VEL_V_MAX_MIN

DESCRIPTION:

This function will find minimum and maximum of 'v' component of linear velocity in *filename* .

SYNTAX:

```
plot_find_vel_v_max_min(plot_vel_v_min,plot_vel_v_max, filename)

char    *filename;
int     *plot_vel_v_min;
int     *plot_vel_v_max;
```

PLOT_HP

DESCRIPTION:

This function will allow the user to plot the data in various files on an HP plotter.

SYNTAX:

plot_hp( )

PLOT_HP_ACC_U

DESCRIPTION:

This function will plot the 'u' component of the linear acceleration of the PRS target(s) (data in *filename*) with respect to time.

SYNTAX:

```
plot_hp_acc_u( filename )

char *filename;
```

PLOT_HP_ACC_UV_LABELS

DESCRIPTION:

This function will plot the labels for the linear acceleration graph. The data is kept in *filename*.

SYNTAX:

```
plot_hp_acc_uv_labels( filename )

char *filename;
```

PLOT_HP_ACC_V

DESCRIPTION:

This function will plot the 'v' component of linear acceleration of the PRS target(s) (data in *filename*) with respect to time.

SYNTAX:

```
plot_hp_acc_v( filename)

char *filename;
```

PLOT_HP_ANG_ACC

DESCRIPTION:

This function will plot the angular acceleration of the PRS targets (data in *filename*) with respect to time.

SYNTAX:

```
plot_hp_ang_acc( filename)

char *filename;
```

PLOT_HP_ANG_ACC_LABELS

DESCRIPTION:

This function will plot the labels for angular acceleration graph. The data is kept in *filename*.

SYNTAX:

```
plot_hp_ang_acc_labels( filename )

char *filename;
```

PLOT_HP_ANG_VEL

DESCRIPTION:

This function will plot the angular velocity of the PRS targets (data in *filename*) with respect to time.

SYNTAX:

```
plot_hp_ang_vel( filename)

char *filename;
```

PLOT_HP_ANG_VEL_LABELS

DESCRIPTION:

This function will plot the labels for angular velocity graph. The data is kept in *filename*.

SYNTAX:

```
plot_hp_ang_vel_labels( filename )

char *filename;
```

PLOT_HP_ANGLE

DESCRIPTION:

This function will plot the orientation of the PRS targets (data in *filename*) with respect to time.

SYNTAX:

plot_hp_angle(filename)

char *filename;

PLOT_HP_ANGLE_LABELS

DESCRIPTION:

This function will plot the labels for the orientation graph. The data is kept in *filename*.

SYNTAX:

```
plot_hp_angle_labels( filename )

char *filename;
```

PLOT_HP_U

DESCRIPTION:

This function will plot the 'u' co-ordinates of the PRS target(s) (data in *filename*) with respect to time.

SYNTAX:

```
plot_hp_u( filename )

char *filename;
```

PLOT_HP_UV_LABELS

DESCRIPTION:

This function will plot the labels for the (u,v) graph. The data is kept in *filename*.

SYNTAX:

```
plot_hp_uv_labels( filename )

char *filename;
```

PLOT_HP_V

DESCRIPTION:

This function will plot the 'v' co-ordinates of the PRS target(s) in *filename* with respect to time.

SYNTAX:

```
plot_hp_v( filename)

char *filename;
```

PLOT_HP_VEL_U

DESCRIPTION:

This function will plot the 'u' component of linear velocity of the PRS target(s) (Data in *filename*) with respect to time.

SYNTAX:

```
plot_hp_vel_u( filename)

char *filename;
```

PLOT_HP_VEL_UV_LABELS

DESCRIPTION:

This function will plot the labels for linear velocity graph. The data is kept in *filename*.

SYNTAX:

```
plot_hp_vel_uv_labels( filename )

char *filename;
```

PLOT_HP_VEL_V

DESCRIPTION:

This function will plot the 'v' component of linear velocity of the PRS target(s) (Data in *filename*) with respect to time.

SYNTAX:

```
plot_hp_vel_v( filename)

char *filename;
```

PLOT_HP_X

DESCRIPTION:

This function will plot the 'X' co-ordinates of the PRS target(s) in *filename* with respect to time.

SYNTAX:

```
plot_hp_x( filename)

char *filename;
```

PLOT_HP_XY_LABELS

DESCRIPTION:

This function will plot the labels for the (X,Y) graph. The data is in *filename*.

SYNTAX:

```
plot_hp_xy_labels( filename)

char *filename;
```

PLOT_HP_Y

DESCRIPTION:

This function will plot the 'Y' co-ordinates of the PRS target(s) in *filename* with respect to time.

SYNTAX:

plot_hp_y(filename)

char *filename;

PRINT_COMMAND

DESCRIPTION:

This function will allow the user to print out data files.

SYNTAX:

print_command()

PRINT_MATRIX

DESCRIPTION:

This function will print out a matrix to a given file stream.

SYNTAX:

```
print_matrix( fp, matrix, row_dim, col_dim )

FILE *fp;
double *matrix;
int row_dim;
int col_dim;
```

PRINT_MENU

DESCRIPTION:

This function will set the menu structure for the menu and supporting text.

SYNTAX:

```
print_menu()
```

PRS

DESCRIPTION:

This is the main routine for PRS.

SYNTAX:

prs

REVISION:

PRS_MAIN_MENU

DESCRIPTION:

This function will set up the main PRS command structure.

SYNTAX:

```
prs_main_menu()
```

READ_IMAGE_FROM_FILE

DESCRIPTION:

This function will read an image from a file. UNSUCCESSFUL (-1) will be returned if unable to read entire file, else SUCCESSFUL (0) will be returned.

SYNTAX:

```
int read_image_from_file( file_name )

char file_name;
```

READ_IMAGE_FROM_MVP

DESCRIPTION:

This function will read an image from the mvp board. UNSUCCESSFUL (-1) will be returned if unable to read entire file, else SUCCESSFUL (0) will be returned.

SYNTAX:

```
int read_image_from_mvp()
```

READ_PIXEL_IN_BUFFER

DESCRIPTION:

This function will return the requested pixel from the HUGE array buffer given its x and y locations.

SYNTAX:

```
int read_pixel_in_buffer( x, y )

int x, y;
```

RECV_COM1

DESCRIPTION:

This function will receive a character from com port 1. The character will be returned as an integer.

SYNTAX:

recv_com1()

RECV_COM2

DESCRIPTION:

This function will receive a character from com port 2. The character will be returned as an integer.

SYNTAX:

recv_com2( )

RECV_STRING1

DESCRIPTION:

This function will receive a character string from serial port 1.

SYNTAX:

recv_string1(in_string)

char *in_string;

RECV_STRING2

DESCRIPTION:

This function will receive a character string from serial port 2.

SYNTAX:

```
recv_string2(in_string)

char *in_string;
```

SEARCH_FOR_TARGET

DESCRIPTION:

This function will search for the robot in the current data in the buffer. Destination should be set to TO_EGA or TO_MVP. If too many threshold pixels are found, the threshold_value will be raised by THRESHOLD_ADJUST_VALUE. Also, if not enough pixels are found, the threshold_value will be reduced by THRESHOLD_ADJUST_VALUE. If the threshold value is raised, lowered or vice versa, the routine will exit UNSUCCESSFUL(ly). Note: If the user hits a key, the search will be interrupted and the keyboard input will be returned in the character array pointed to by the variable keyboard.

SYNTAX:

```
search_for_target( destination, original_threshold, keyboard )

int destination;
int original_threshold;
char *keyboard;
```

SEND_COM1

DESCRIPTION:

This function will send a character to com (serial) port 1.

SYNTAX:

send_com1(send_char)

char send_char;

SEND_COM2

DESCRIPTION:

This function will send a character to com (serial) port 2.

SYNTAX:

```
send_com2(send_char)

char send_char;
```

SEND_STRING1

DESCRIPTION:

This function will send a character string to com port 1.

SYNTAX:

```
send_string1(command)

char *command;
```

SEND_STRING2

DESCRIPTION:

This function will send a character string to com port 2.

SYNTAX:

```
send_string2(command)

char *command;
```

SLOW_AT

DESCRIPTION:

This function will slow the PC's Limited 286 to 6 Mz. It is not required if Compaq's 386/20 is used.

SYNTAX:

slow_at()

START_STOP_WATCH

DESCRIPTION:

This function will start the stop-watch time.

SYNTAX:

start_stop_watch( )

STAT_COM1

DESCRIPTION:

This function will return the status of com port 1 to the caller.

SYNTAX:

stat_com1()

STAT_COM2

DESCRIPTION:

This function will return the status of com port 2 to the caller.

SYNTAX:

stat_com2()

SW_CAMERA

DESCRIPTION:

This function will switch the VCS Video Switcher to the *chan_no* camera.

SYNTAX:

```
sw_camera(chan_no)

int chan_no;
```

TRACK

DESCRIPTION:

This function will track the PRS targets.

SYNTAX:

track()

TRACK_MAP

DESCRIPTION:

This function will draw a map of the camera floor layout which is a representation of the non-overlapping FOV of all the cameras.

SYNTAX:

track_map( )

TRACK_SNAP_SHOT

DESCRIPTION:

This function will read a rectangular image from the frame grabber board with the center being *x_target*, *y_target* and the x and y offset from the center to the edges being TRACK_X_OFFSET and TRACK_Y_OFFSET. The data is placed in the appropriate locations in the buffer. Note: No boundary checking is done. If the target is too close to the edge, garbage data will be returned.

SYNTAX:

```
track_snap_shot( x_target, y_target )

int x_target, y_target;
```

UTIL_MENU

DESCRIPTION:

This function will set the menu structure for the utility command window.

SYNTAX:

util_menu()

UTILITIES

DESCRIPTION:

This function will allow the user to modify the profile settings.

SYNTAX:

int utilities()

VERIFY_TARGET

DESCRIPTION:

This function will verify that the targets which were found correspond to the desired target. If this the case, the PRS targets (LEDs) will be numbered in the following order:

TARGET
L1 L3

L2

If we are able to verify the target, we will return SUCCESSFUL (0).

SYNTAX:

int verify_target( )

XY_MATRIX

DESCRIPTION:

This function will set up the (X,Y) matrix.

SYNTAX:

```
xy_matrix( dx, dy, dm )

double   dx, dy, *dm;
```

SECTION C

LIBRARY REFERENCE MANUAL

ANY_KEY

DESCRIPTION:

This `void` function waits for a key to be pressed, then returns.

SYNTAX:

```
any_key();
```

BEEP

DESCRIPTION:

This function will produce a beep.

SYNTAX:

```
beep();
```

BOX_SCREEN

DESCRIPTION:

This function will place a box on the screen.

SYNTAX:

box_screen(row, column, height, width, attribute)

int row, column, height, width, attribute;

CHKKEY

DESCRIPTION:

This int function will return a 0 if a keystroke is ready and the key plus scan code will be returned in the parameter list. If no keystroke is ready a non-zero result will be returned.

SYNTAX:

```
char scan_code[2];
int chkkey( scan_code)
```

EXAMPLE:

```
char scan_code[2];
int chkkey();

if (chkkey( scan_code) == 0)
printf("the following scancode was returned: %c, %c",
        scan_code, scan_code + 1);
```

COMMAND_CLEAR

DESCRIPTION:

This `void` function will clear the commands from the screen. The command structure must be set by using command_set.

SYNTAX:

```
void command_clear();
```

EXAMPLE:

```
command_clear();
```

COMMAND_HANDLER

DESCRIPTION:

This `void` function will take a scancode and handle any command structure movement and management necessary. The command structure must have been set up previously by command_set.

SYNTAX:

```
char scan_code[2];
void command_handler( scan_code)
```

EXAMPLE:

```
char scan_code[2];

inkey( scan_code);
command_handler( scan_code);
```

COMMAND_INIT

DESCRIPTION:

This `void` function will place the command structure on the screen. The command structure must have been set up previously by command_set.

SYNTAX:

`void` command_init()

EXAMPLE:

```
command_init();
```

COMMAND_PARAMETER_SET

DESCRIPTION:

This `void` function will set up the command parameter structure. This will set the number of commands, highlighted attribute, normal attribute, current command, and control direction.

SYNTAX:

```
int number, highlighted_attribute, normal_attribute;
int current_command, control_direction;

void command_parameter_set( number, highlighted_attribute,
normal_attribute, current_command, control_direction)
```

EXAMPLE:

```
#include "include\command.ext"
#define GRAY 0x07
#define REVGRAY 0x70

command_parmameter_set( 5, GRAY, REVGRAY, 0,
                                    RIGHT_LEFT_CONTROL);
```

COMMAND_SET

DESCRIPTION:

This `void` function will set up the command structure for each command. command_set will set the row, column, and string label for a given command of the structure.

SYNTAX:

```
int row, column;
char label[];
void command_set(row, column, label)
```

EXAMPLE:

```
int row = 5, column = 20;
char label[6] = "Hello";

command_set( row, column, label);
```

FKEY_BLOCK

DESCRIPTION:

This `void` function places function key labels on the CRT.

SYNTAX:

```
void fkey_block(fkeys)
char *fkeys[10];
```

SEMANTICS:

Function key numbers are placed on the screen in inverse video. All labels should be initialized; even those which are to remain empty. The routine erases any characters which were in the function key area before printing out the new keys.

EXAMPLE:

```
char *fkey_set_2[10];

fkey_set_2[0] = "Key 1";
fkey_set_2[1] = "KEY 2";
```

```
fkey_set_2[2] = "KEY 3";
fkey_set_2[3] = "KEY 4";
fkey_set_2[4] = "KEY 5";
fkey_set_2[5] = "KEY 6";
fkey_set_2[6] = "KEY 7";
fkey_set_2[7] = "KEY 8";
fkey_set_2[8] = "KEY 9";
fkey_set_2[9] = "KEY 10";

fkey_block(fkey_set_1);
```

FKEY_CLEAR

DESCRIPTION:

This `void` function erases the function key labels.

SYNTAX:

```
void fkey_clear();
```

EXAMPLE:

```
fkey_clear();
```

FLOOD_PARTIAL_SCREEN

DESCRIPTION:

This function will flood the partial screen with a given character and a given attribute.

SYNTAX:

```
int row, column, height, width, attribute;
char flood_character;

flood_partial_screen( row, column, height, width,
flood_character, attribute)
```

INKEY

DESCRIPTION:

This `void` function will wait for a key to be hit for on keyboard and will return the scancode without echoing the character to the screen.

SYNTAX:

```
char scan_code[2];
void inkey( scan_code)
```

EXAMPLE:

```
char scan_code[2];
void inkey();
```

INTEGER_BOUNDRY

DESCRIPTION:

This integer fuction will force an integer to be within a maximum and minimum boundary.

SYNTAX:

```
int integer_boundry( value, max_value, min_value)

int value, max_value, min_value;
```

SEMANTICS:

The integer value is checked against the max_value and if it is greater than the function, returns max_value. If value is less than min_value than the function will return min_value. Otherwise the function will return value.

MENU_ALLOCATE

DESCRIPTION:

This `void` function will set up temporary character fields for the menu structure. This must be called before the menu is used. Use menu_retrieve to transfer the information from the temporary fields to the original data fields.

SYNTAX:

```
void menu_allocate();
```

EXAMPLE:

```
menu_allocate();
```

MENU_ATTR

DESCRIPTION:

This `void` function places an attribute throughout all the fields of a menu.

SYNTAX:

```
menu_attr( attr)
 char attr;
```

EXAMPLE:

```
menu_attr( 0x7);
```

MENU_ATTR_FIELD

DESCRIPTION:

This void function will attribute an individual field.

SYNTAX:

```
int field, attr;
void menu_attr_field( field, attr);
```

EXAMPLE:

```
#define GRAY 0x7
menu_attr_field( 0, GRAY);
```

MENU_CHECK

DESCRIPTION:

This void function will check the field of a menu and return a 0 if all the fields are valid, otherwise it returns a non-zero value and a bad field number.

SYNTAX:

```
int *invalid_field;
void menu_check( invalid_field)
```

EXAMPLE:

```
menu_check( invalid_field);
```

MENU_CLEAR_FIELD

DESCRIPTION:

This `void` function will clear a given field with NULLs.

SYNTAX:

```
int field;
void menu_clear_field(field)
```

EXAMPLE:

```
menu_clear_field(0);
```

MENU_CLR

DESCRIPTION:

This `void` function will clear all the fields in a menu structure.

SYNTAX:

```
char character;
void menu_clr( character)
```

EXAMPLE:

```
char character = ' ';
menu_clr( character);
```

MENU_CURSOR

DESCRIPTION:

This `void` function will update the cursor to the current menu field.

SYNTAX:

```
void menu_cursor();
```

EXAMPLE:

```
menu_cursor();
```

MENU_FREE

DESCRIPTION:

This `void` function will deallocate the memory which was dynamically allocated for the temporary character fields with the routine menu_allocate.

SYNTAX:

```
void menu_free();
```

EXAMPLE:

```
menu_free();
```

MENU_HANDLER

DESCRIPTION:

This function will handle all menu input and display to the screen. The menu structure must be set up with menu_set. The routine menu_handler will return the following codes. Return codes: MENU_NO_ERROR = (0) successful MENU_LIMIT_ERROR = (1) field out of limit

SYNTAX:

```
char *c;
void menu_handler( c)
```

EXAMPLE:

```
char c[2];
inkey( c);
menu_handler( c);
```

MENU_INIT

DESCRIPTION:

This `void` function will initialize a menu structure to the screen.

SYNTAX:

```
void menu_init();
```

EXAMPLE:

```
menu_init();
```

REVISION:

09-14-87 MWP Removed \0 from the sprintf statements.
Remove the redundant memset calls.
Create menu_set_field().

MENU_INIT_FIELD

DESCRIPTION:

This `void` function will initialize a particular field.

SYNTAX:

```
int field;
void menu_init_field(field)
```

EXAMPLE:

```
menu_init_field(0);
```

MENU_INIT_FIELD_ATTR

DESCRIPTION:

This `void` function will initialize a field with a given attribute.

SYNTAX:

```
int field;
char attr;
void menu_init_field_attr( field, attr)
```

EXAMPLE:

```
#define GRAY 0x7
menu_init_field_attr( 0, GRAY);
```

MENU_RETRIEVE

DESCRIPTION:

This `void` function will retrieve the menu structure data from the temporary character strings to the appropriate data fields.

SYNTAX:

```
void menu_retrieve();
```

EXAMPLE:

```
menu_retrieve();
```

MENU_SET

DESCRIPTION:

This `void` function will set the menu structure for a particular field.

SYNTAX:

```
int field_number;
int column_start;
int row_location;
int field_length;
char *home;
int right;
int left;
int down;
int up;
int type;
int num_dec;
int check_limit;
double lower;
double upper;
```

```
int format;
int check_format;
double empty_value;
int display;
char *enumerated_data_fields;
int num_of_enumerated_types;
int length_of_enumerated_field;
menu_set(field_number, column_start, row_location,
field_length, home, right, left, down, up, type,
num_dec, check_limit, lower, upper, format,
check_format, empty_value, display,
enumerated_data_types,
num_of_enumerated_types,
length_of_enumerated_field)
```

SEMANTICS:

The following fields need further explanation. The values of the constants have been defined in *menu.h*.

int type; May assume the following values:
STRING_TYPE 0 INT_TYPE 1 DOUBLE_TYPE 2
FILE_NAME_TYPE 3 CAP_STRING_TYPE 4 ENUMERATED_TYPE 5
LONG_TYPE 6

int format; May assume the following values:
NO_FORMAT 0 LEFT_FORMAT 1 RIGHT_FORMAT 2
CENTER_FORMAT 3

int check_format; May assume the following values:
BLANK_OK 0 FILLED_IN 1

int check_limit; May assume the following values:
NO_LIMIT_CHECK 0 LIMIT_CHECK 1

int display; May assume the following values:
ALWAYS 0
IF_NOT_EMPTY 1

MENU_SET_FIELD

DESCRIPTION:

This `void` function will set the menu structure.

SYNTAX:

```
void menu_set_field();
```

REVISION:

MESSAGE

DESCRIPTION:

This function will print a message to the message line on the screen.

SYNTAX:

```
message( string )

char *string;
```

PRINT

DESCRIPTION:

This `void` function will print a given string to the current cursor location an the screen with a given attribute.

SYNTAX:

```
print(string, attribute);

char *string;
int attribute;
```

SEMANTICS:

This function will write to the screen in one of three ways depending on the value of screen.use_bios. The three methods are BIOS, DIRECT, and PGA_DISPLAY. These values and the `global` variables necessary are in *screen.ext*. The values for the attribute can be found in *color.def*.

EXAMPLE:

```
print( "this is a test", BLUE);
```

PRINT_CHARACTER

DESCRIPTION:

This function will set up a string of number of characters which will be printed with a given attribute.

SYNTAX:

```
char c;
int count, attribute;

print_character( c, count, attribute)
```

PRINT_NO_ATTR

DESCRIPTION:

This void function will print a given string to the current cursor location on the screen with the current attribute.

SYNTAX:

```
print_no_attr( string)

char *string;
```

SEMANTICS:

This function will write to the screen in one of three ways depending on the value of screen.use_bios. The three methods are BIOS, DIRECT, and PGA_DISPLAY. These values and the `global` variables necessary are in *screen.ext*.

EXAMPLE:

```
print_no_attr(" this is a test");
```

QUERY_MESSAGE

DESCRIPTION:

This function will print a query message to the screen and the user will be given a chance to input Y or N. If a Y is entered, the function will return (0); else (1) will be returned.

SYNTAX:

```
int query_message( string )

char *string;
```

QUIT

DESCRIPTION:

This function will clean up the screen before leaving a program.

SYNTAX:

```
quit(rc)

int rc;
```

REV

DESCRIPTION:

Converts a color to its reverse video value.

SYNTAX:

```
rev(color);

int color;
```

SEMANTICS:

Useful for specifying color to other library routines, such as menu_han.

EXAMPLE:

```
highlight = rev(GRAY);
```

SCR_CLR

DESCRIPTION:

This `void` function clears the screen.

SYNTAX:

```
scr_clr();
```

SEMANTICS:

This function requires some `global` variables which are in *screen.ext.*

EXAMPLE:

```
scr_clr();
```

SCR_CURSOFF

DESCRIPTION:

This `void` function turns the cursor *off*.

SYNTAX:

```
scr_cursoff();
```

SEMANTICS:

This function requires some `global` variables which are in *screen.ext*.

EXAMPLE:

```
scr_cursoff();
```

SCR_CURSON

DESCRIPTION:

This `void` function turns the cursor *on.*

SYNTAX:

```
scr_cursoff();
```

SEMANTICS:

This function requires some `global` variables which are in *screen.ext.*

EXAMPLE:

```
scr_curson();
```

SCR_PAGE

DESCRIPTION:

This `void` function will set the screen default page.

SYNTAX:

```
scr_page( page)

int page;
```

SEMANTICS:

This function requires some `global` variables which are in *screen.ext.*

EXAMPLE:

```
scr_page(0);
```

SCR_POS

DESCRIPTION:

This `void` function will return the current cursor location.

SYNTAX:

```
scr_pos( old_row, old_column)

int *old_row, *old_column;
```

SEMANTICS:

This function requires some `global` variables which are in *screen.ext.*

EXAMPLE:

```
int old_row, old_column;

scr_pos( &old_row, &old_column);
```

SCR_ROWCOL

DESCRIPTION:

This `void` function will set the current cursor location.

SYNTAX:

```
scr_rowcol( row, column)

int row, column;
```

SEMANTICS:

This function requires some `global` variables which are in *screen.ext.*

EXAMPLE:

```
scr_pos( 5, 10);
```

SCR_SETUP

DESCRIPTION:

This `void` function sets up the `global` screen parameters.

SYNTAX:

```
scr_setup();
```

SEMANTICS:

This routine must be called before any scr_ routines are used. Also, *screen.h* must be included in the main which defines the necessary `global` variables.

EXAMPLE:

```
scr_setup();
```

WRTATTR

DESCRIPTION:

This `void` function will write a given attibute to a string of characters on the screen located at the current cursor position.

SYNTAX:

```
wrtattr( attribute)

int attribute;
```

SEMANTICS:

This function will write to the screen in one of three ways depending on the value of screen.use_bios. The three methods are BIOS, DIRECT, and PGA_DISPLAY. These values and the global variables necessary are in *screen.ext*. The values for the attribute can be found in *color.def*.

EXAMPLE:

```
wrtattr( BLUE);
```

BIBLIOGRAPHY

[1] Position Reference System - A User's Manual.