```
;-------------------------------------------------------------------------------
;-------------------------------------------------------------------------------
FUNCTION IMDISP_GETPOS, ASPECT, POSITION=POSITION, MARGIN=MARGIN

;- Compute a position vector given an aspect ratio (called by IMDISP_IMSIZE)

;- Check arguments
if (n_params() ne 1) then message, 'Usage: RESULT = IMDISP_GETPOS(ASPECT)'
if (n_elements(aspect) eq 0) then message, 'ASPECT is undefined'

;- Check keywords
if (n_elements(position) eq 0) then position = [0.0, 0.0, 1.0, 1.0]
if (n_elements(margin) eq 0) then margin = 0.1

;- Get range limited aspect ratio and margin input values
aspect_val = (float(aspect[0]) > 0.01) < 100.0
margin_val = (float(margin[0]) > 0.0) < 0.495

;- Compute aspect ratio of position vector in this window
xsize = (position[2] - position[0]) * !d.x_vsize
ysize = (position[3] - position[1]) * !d.y_vsize
cur_aspect = ysize / xsize

;- Compute aspect ratio of this window
win_aspect = float(!d.y_vsize) / float(!d.x_vsize)

;- Compute height and width in normalized units
if (aspect_val ge cur_aspect) then begin
  height = (position[3] - position[1]) - 2.0 * margin
  width  = height * (win_aspect / aspect_val)
endif else begin
  width  = (position[2] - position[0]) - 2.0 * margin
  height = width * (aspect_val / win_aspect)
endelse

;- Compute and return position vector
xcenter = 0.5 * (position[0] + position[2])
ycenter = 0.5 * (position[1] + position[3])
x0 = xcenter - 0.5 * width
y0 = ycenter - 0.5 * height
x1 = xcenter + 0.5 * width
y1 = ycenter + 0.5 * height
return, [x0, y0, x1, y1]

END
;-------------------------------------------------------------------------------
FUNCTION IMDISP_IMSCALE, IMAGE, RANGE=RANGE, BOTTOM=BOTTOM, NCOLORS=NCOLORS, $
  NEGATIVE=NEGATIVE

;- Byte-scale an image (called by IMDISP)

;- Check arguments
if (n_params() ne 1) then message, 'Usage: RESULT = IMDISP_IMSCALE(IMAGE)'
if (n_elements(image) eq 0) then message, 'Argument IMAGE is undefined'

;- Check keywords
if (n_elements(range) eq 0) then begin
  min_value = min(image, max=max_value)
  range = [min_value, max_value]
endif
if (n_elements(bottom) eq 0) then bottom = 0B
if (n_elements(ncolors) eq 0) then ncolors = !d.table_size - bottom

;- Compute the scaled image
scaled = bytscl(image, min=range[0], max=range[1], top=(ncolors - 1))

```

```
;- Create a negative image if required
if keyword_set(negative) then scaled = byte(ncolors - 1) - scaled

;- Return the scaled image in the correct color range
return, scaled + byte(bottom)

END
;------------------------------------------------------------------------------
FUNCTION IMDISP_IMREGRID, DATA, NX, NY, INTERP=INTERP

;- Regrid a 2D array (called by IMDISP)

;- Check arguments
if (n_params() ne 3) then $
  message, 'Usage: RESULT = IMDISP_IMREGRID(DATA, NX, NY)'
if (n_elements(data) eq 0) then message, 'Argument DATA is undefined'
result = size(data)
ndims = result[0]
dims = result[1:ndims]
if (ndims ne 2) then message, 'Argument DATA must have 2 dimensions'
if (n_elements(nx) eq 0) then message, 'Argument NX is undefined'
if (n_elements(ny) eq 0) then message, 'Argument NY is undefined'
if (nx lt 1) then message, 'NX must be 1 or greater'
if (ny lt 1) then message, 'NY must be 1 or greater'

;- Copy the array if the requested size is the same as the current size
if (nx eq dims[0]) and (ny eq dims[1]) then begin
  new = data
  return, new
endif

;- Compute index arrays for bilinear interpolation
xindex = (findgen(nx) + 0.5) * (dims[0] / float(nx)) - 0.5
yindex = (findgen(ny) + 0.5) * (dims[1] / float(ny)) - 0.5

;- Round the index arrays if nearest neighbor sampling is required
if (keyword_set(interp) eq 0) then begin
  xindex = round(xindex)
  yindex = round(yindex)
endif

;- Return regridded array
return, interpolate(data, xindex, yindex, /grid)

END
;------------------------------------------------------------------------------
PRO IMDISP_IMSIZE, IMAGE, X0, Y0, XSIZE, YSIZE, ASPECT=ASPECT, $
  POSITION=POSITION, MARGIN=MARGIN

;- Compute the size and offset for an image (called by IMDISP)

;- Check arguments
if (n_params() ne 5) then $
  message, 'Usage: IMDISP_IMSIZE, IMAGE, X0, Y0, XSIZE, YSIZE'
if (n_elements(image) eq 0) then $
  message, 'Argument IMAGE is undefined'
if (n_elements(position) eq 0) then position = [0.0, 0.0, 1.0, 1.0]
if (n_elements(position) ne 4) then $
  message, 'POSITION must be a 4 element vector'
if (n_elements(margin) eq 0) then margin = 0.1
if (n_elements(margin) ne 1) then $
  message, 'MARGIN must be a scalar'

;- Get image dimensions
result = size(image)
ndims = result[0]
```

```
133 if (ndims ne 2) then message, 'IMAGE must be a 2D array'
134 dims = result[1 : ndims]
135
136 ;- Get aspect ratio for image
137 if (n_elements(aspect) eq 0) then $
138   aspect = float(dims[1]) / float(dims[0])
139 if (n_elements(aspect) ne 1) then $
140   message, 'ASPECT must be a scalar'
141
142 ;- Check output parameters
143 if (arg_present(x0) ne 1) then message, 'Argument XO cannot be set'
144 if (arg_present(y0) ne 1) then message, 'Argument YO cannot be set'
145 if (arg_present(xsize) ne 1) then message, 'Argument XSIZE cannot be set'
146 if (arg_present(ysize) ne 1) then message, 'Argument YSIZE cannot be set'
147
148 ;- Get approximate image position
149 position = imdisp_getpos(aspect, position=position, margin=margin)
150
151 ;- Compute lower left position of image (device units)
152 x0 = round(position[0] * !d.x_vsize) > 0L
153 y0 = round(position[1] * !d.y_vsize) > 0L
154
155 ;- Compute size of image (device units)
156 xsize = round((position[2] - position[0]) * !d.x_vsize) > 2L
157 ysize = round((position[3] - position[1]) * !d.y_vsize) > 2L
158
159 ;- Recompute the image position based on actual image size
160 position = fltarr(4)
161 position[0] = x0 / float(!d.x_vsize)
162 position[1] = y0 / float(!d.y_vsize)
163 position[2] = (x0 + xsize) / float(!d.x_vsize)
164 position[3] = (y0 + ysize) / float(!d.y_vsize)
165
166 END
167 ;-----------------------------------------------------------------------------
168 PRO DISPL, IMAGE, RANGE=RANGE, BOTTOM=BOTTOM, NCOLORS=NCOLORS, $
169   MARGIN=MARGIN, INTERP=INTERP, DITHER=DITHER, ASPECT=ASPECT, $
170   POSITION=POSITION, OUT_POS=OUT_POS, NOSCALE=NOSCALE, NORESIZE=NORESIZE, $
171   ORDER=ORDER, USEPOS=USEPOS, CHANNEL=CHANNEL, $
172   BACKGROUND=BACKGROUND, ERASE=ERASE, $
173   AXIS=AXIS, NEGATIVE=NEGATIVE, _EXTRA=EXTRA_KEYWORDS
174
175 ;+
176 ; NAME:
177 ;    IMDISP
178 ;
179 ; PURPOSE:
180 ;    Display an image on the current graphics device.
181 ;    IMDISP is an advanced replacement for TV and TVSCL.
182 ;
183 ;    - Supports WIN, MAC, X, CGM, PCL, PRINTER, PS, and Z graphics devices,
184 ;    - Image is automatically byte-scaled (can be disabled),
185 ;    - Custom byte-scaling of Pseudo color images via the RANGE keyword,
186 ;    - Pseudo (indexed) color and True color images are handled automatically,
187 ;    - 8-bit and 24-bit graphics devices  are handled automatically,
188 ;    - Decomposed color settings are handled automatically,
189 ;    - Image is automatically sized to fit the display (can be disabled),
190 ;    - The !P.MULTI system variable is honored for multiple image display,
191 ;    - Image can be positioned via the POSITION keyword,
192 ;    - Color table splitting via the BOTTOM and NCOLORS keywords,
193 ;    - Image aspect ratio customization via the ASPECT keyword,
194 ;    - Resized images can be resampled (default) or interpolated,
195 ;    - Top down image display via the ORDER keyword (!ORDER is ignored),
196 ;    - Selectable display channel (R/G/B) via the CHANNEL keyword,
197 ;    - Background can be set to a specified color via the BACKGROUND keyword,
198 ;    - Screen can be erased prior to image display via the ERASE keyword,
```

```
;    - Plot axes can be drawn on the image via the AXIS keyword,
;    - Photographic negative images can be displayed via the NEGATIVE keyword.
;
; CATEGORY:
;    Image display
;
; CALLING SEQUENCE:
;    IMDISP, IMAGE
;
; INPUTS:
;    IMAGE       Array containing image data.
;                Pseudo (indexed) color images must have 2 dimensions.
;                True color images must have 3 dimensions, in either
;                [3, NX, NY], [NX, 3, NY], or [NX, NY, 3] form.
;
; OPTIONAL INPUTS:
;    None.
;
; KEYWORD PARAMETERS:
;    RANGE       For Pseudo Color images only, a vector with two elements
;                specifying the minimum and maximum values of the image
;                array to be considered when the image is byte-scaled
;                (default is minimum and maximum array values).
;                This keyword is ignored for True Color images,
;                or if the NOSCALE keyword is set.
;
;    BOTTOM      Bottom value in the color table to be used
;                for the byte-scaled image
;                (default is 0).
;                This keyword is ignored if the NOSCALE keyword is set.
;
;    NCOLORS     Number of colors in the color table to be used
;                for the byte-scaled image
;                (default is !D.TABLE_SIZE - BOTTOM).
;                This keyword is ignored if the NOSCALE keyword is set.
;
;    MARGIN      A scalar value specifying the margin to be maintained
;                around the image in normal coordinates
;                (default is 0.1, or 0.025 if !P.MULTI is set to display
;                multiple images).
;
;    INTERP      If set, the resized image will be interpolated using
;                bilinear interpolation
;                (default is nearest neighbor sampling).
;
;    DITHER      If set, true color images will be dithered when displayed
;                on an 8-bit graphics device
;                (default is no dithering).
;
;    ASPECT      A scalar value specifying the aspect ratio (height/width)
;                for the displayed image
;                (default is to maintain native aspect ratio).
;
;    POSITION    On input, a 4-element vector specifying the position
;                of the displayed image in the form [X0,Y0,X1,Y1] in
;                in normal coordinates
;                (default is [0.0,0.0,1.0,1.0]).
;                See the examples below to display an image where only the
;                offset and size are known (e.g. MAP_IMAGE output).
;
;    OUT_POS     On output, a 4-element vector specifying the position
;                actually used to display the image.
;
;    NOSCALE     If set, the image will not be byte-scaled
;                (default is to byte-scale the image).
;
```

```
265 ;    NORESIZE    If set, the image will not be resized.
266 ;                (default is to resize the image to fit the display).
267 ;
268 ;    ORDER       If set, the image is displayed from the top down
269 ;                (default is to display the image from the bottom up).
270 ;                Note that the system variable !ORDER is always ignored.
271 ;
272 ;    USEPOS      If set, the image will be sized to exactly fit a supplied
273 ;                POSITION vector, over-riding ASPECT and MARGIN
274 ;                (default is to honor ASPECT and MARGIN when a POSITION
275 ;                vector is supplied).
276 ;
277 ;    CHANNEL     Display channel (Red, Green, or Blue) to be written.
278 ;                0 => All channels (the default)
279 ;                1 => Red channel
280 ;                2 => Green channel
281 ;                3 => Blue channel
282 ;                This keyword is only recognized by graphics devices which
283 ;                support 24-bit decomposed color (WIN, MAC, X). It is ignored
284 ;                by all other graphics devices. However True color (RGB)
285 ;                images can be displayed on any device supported by IMDISP.
286 ;
287 ;    BACKGROUND  If set to a positive integer, the background will be filled
288 ;                with the color defined by BACKGROUND.
289 ;
290 ;    ERASE       If set, the screen contents will be erased. Note that if
291 ;                !P.MULTI is set to display multiple images, the screen is
292 ;                always erased when the first image is displayed.
293 ;
294 ;    AXIS        If set, plot axes will be drawn on the image. The default
295 ;                x and y axis ranges are determined by the size of the image.
296 ;                When the AXIS keyword is set, IMDISP accepts any keywords
297 ;                supported by PLOT (e.g. TITLE, COLOR, CHARSIZE etc.).
298 ;
299 ;    NEGATIVE    If set, a photographic negative of the image is displayed.
300 ;                The values of BOTTOM and NCOLORS are honored. This keyword
301 ;                allows True color images scanned from color negatives to be
302 ;                displayed. It also allows Pseudo color images to be displayed
303 ;                as negatives without reversing the color table. This keyword
304 ;                is ignored if the NOSCALE keyword is set.
305 ;
306 ; OUTPUTS:
307 ;    None.
308 ;
309 ; OPTIONAL OUTPUTS:
310 ;    None
311 ;
312 ; COMMON BLOCKS:
313 ;    None
314 ;
315 ; SIDE EFFECTS:
316 ;    The image is displayed on the current graphics device.
317 ;
318 ; RESTRICTIONS:
319 ;    Requires IDL 5.0 or higher (square bracket array syntax).
320 ;
321 ; EXAMPLE:
322 ;
323 ;;- Load test data
324 ;
325 ;openr, lun, filepath('ctscan.dat', subdir='examples/data'), /get_lun
326 ;ctscan = bytarr(256, 256)
327 ;readu, lun, ctscan
328 ;free_lun, lun
329 ;openr, lun, filepath('hurric.dat', subdir='examples/data'), /get_lun
330 ;hurric = bytarr(440, 330)
```

```
;readu, lun, hurric
;free_lun, lun
;read_jpeg, filepath('rose.jpg', subdir='examples/data'), rose
;help, ctscan, hurric, rose
;
;;- Display single images
;
;!p.multi = 0
;loadct, 0
;imdisp, hurric, /erase
;wait, 3.0
;imdisp, rose, /interp, /erase
;wait, 3.0
;
;;- Display multiple images without color table splitting
;;- (works on 24-bit displays only; top 2 images are garbled on 8-bit displays)
;
;!p.multi = [0, 1, 3, 0, 0]
;loadct, 0
;imdisp, ctscan, margin=0.02
;loadct, 13
;imdisp, hurric, margin=0.02
;imdisp, rose, margin=0.02
;wait, 3.0
;
;;- Display multiple images with color table splitting
;;- (works on 8-bit or 24-bit displays)
;
;!p.multi = [0, 1, 3, 0, 0]
;loadct, 0, ncolors=64, bottom=0
;imdisp, ctscan, margin=0.02, ncolors=64, bottom=0
;loadct, 13, ncolors=64, bottom=64
;imdisp, hurric, margin=0.02, ncolors=64, bottom=64
;imdisp, rose, margin=0.02, ncolors=64, bottom=128
;wait, 3.0
;
;;- Display an image at a specific position, over-riding aspect and margin
;
;!p.multi = 0
;loadct, 0
;imdisp, hurric, position=[0.0, 0.0, 1.0, 0.5], /usepos, /erase
;wait, 3.0
;
;;- Display an image with axis overlay
;
;!p.multi = 0
;loadct, 0
;imdisp, rose, /axis, /erase
;wait, 3.0
;
;;- Display an image with contour plot overlay
;
;!p.multi = 0
;loadct, 0
;imdisp, hurric, out_pos=out_pos, /erase
;contour, smooth(hurric, 10, /edge), /noerase, position=out_pos, $
;  xstyle=1, ystyle=1, levels=findgen(5)*40.0, /follow
;wait, 3.0
;
;;- Display a small image with correct resizing
;
;!p.multi = 0
;loadct, 0
;data = (dist(8))[1:7, 1:7]
;imdisp, data, /erase
;wait, 3.0
```

```
;imdisp, data, /interp
;wait, 3.0
;
;;- Display a true color image without and with interpolation
;
;!p.multi = 0
;imdisp, rose, /erase
;wait, 3.0
;imdisp, rose, /interp
;wait, 3.0
;
;;- Display a true color image as a photographic negative
;
;imdisp, rose, /negative, /erase
;wait, 3.0
;
;;- Display a true color image on PostScript output
;;- (note that color table is handled automatically)
;
;current_device = !d.name
;set_plot, 'PS'
;device, /color, bits_per_pixel=8, filename='imdisp_true.ps'
;imdisp, rose, /axis, title='PostScript True Color Output'
;device, /close
;set_plot, current_device
;
;;- Display a pseudo color image on PostScript output
;
;current_device = !d.name
;set_plot, 'PS'
;device, /color, bits_per_pixel=8, filename='imdisp_pseudo.ps'
;loadct, 0
;imdisp, hurric, /axis, title='PostScript Pseudo Color Output'
;device, /close
;set_plot, current_device
;
;;- Display an image where only the offset and size are known
;
;;- Read world elevation data
;file = filepath('worldelv.dat', subdir='examples/data')
;openr, lun, file, /get_lun
;data = bytarr(360, 360)
;readu, lun, data
;free_lun, lun
;;- Reorganize array so it spans 180W to 180E
;world = data
;world[0:179, *] = data[180:*, *]
;world[180:*, *] = data[0:179, *]
;;- Create remapped image
;map_set, /orthographic, /isotropic, /noborder
;remap = map_image(world, x0, y0, xsize, ysize, compress=1)
;;- Convert offset and size to position vector
;pos = fltarr(4)
;pos[0] = x0 / float(!d.x_vsize)
;pos[1] = y0 / float(!d.y_vsize)
;pos[2] = (x0 + xsize) / float(!d.x_vsize)
;pos[3] = (y0 + ysize) / float(!d.y_vsize)
;;- Display the image
;loadct, 0
;imdisp, remap, pos=pos, /usepos
;map_continents
;map_grid
;
; MODIFICATION HISTORY:
; Liam.Gumley@ssec.wisc.edu
; http://cimss.ssec.wisc.edu/~gumley
```

```
; $Id: imdisp.pro,v 1.45 2000/08/28 16:17:14 gumley Exp $
;
; Copyright (C) 1999, 2000 Liam E. Gumley
;
; This program is free software; you can redistribute it and/or
; modify it under the terms of the GNU General Public License
; as published by the Free Software Foundation; either version 2
; of the License, or (at your option) any later version.
;
; This program is distributed in the hope that it will be useful,
; but WITHOUT ANY WARRANTY; without even the implied warranty of
; MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
; GNU General Public License for more details.
;
; You should have received a copy of the GNU General Public License
; along with this program; if not, write to the Free Software
; Foundation, Inc., 59 Temple Place - Suite 330, Boston, MA  02111-1307, USA.
;-

rcs_id = '$Id: imdisp.pro,v 1.45 2000/08/28 16:17:14 gumley Exp $'

;----------------------------------------------------------------------------
;- CHECK INPUT
;----------------------------------------------------------------------------

;- Check arguments
if (n_params() ne 1) then message, 'Usage: IMDISP, IMAGE'
if (n_elements(image) eq 0) then message, 'Argument IMAGE is undefined'
if (max(!p.multi) eq 0) then begin
  if (n_elements(margin) eq 0) then begin
    if (n_elements(position) eq 4) then margin = 0.0 else margin = 0.1
  endif
endif else begin
  if (n_elements(margin) eq 0) then margin = 0.025
endelse
if (n_elements(order) eq 0) then order = 0
if (n_elements(channel) eq 0) then channel = 0

;- Check position vector
if (n_elements(position) gt 0) then begin
  if (n_elements(position) ne 4) then $
    message, 'POSITION must be a 4 element vector of the form [X0, Y0, X1, Y1]'
  if (position[0] lt 0.0) then message, 'POSITION[0] must be GE 0.0'
  if (position[1] lt 0.0) then message, 'POSITION[1] must be GE 0.0'
  if (position[2] gt 1.0) then message, 'POSITION[2] must be LE 1.0'
  if (position[3] gt 1.0) then message, 'POSITION[3] must be LE 1.0'
  if (position[0] ge position[2]) then $
    message, 'POSITION[0] must be LT POSITION[2]'
  if (position[1] ge position[3]) then $
    message, 'POSITION[1] must be LT POSITION[3]'
endif

;- Check the image dimensions
result = size(image)
ndims = result[0]
if (ndims lt 2) or (ndims gt 3) then $
  message, 'IMAGE must be a Pseudo Color (2D) or True Color (3D) image array'
dims = result[1:ndims]

;- Check that 3D image array is in valid true color format
true = 0
if (ndims eq 3) then begin
  index = where(dims eq 3L, count)
  if (count eq 0) then $
    message, 'True Color dimensions must be [3,NX,NY], [NX,3,NY], or [NX,NY,3]'
  true = 1
```

```
  truedim = index[0]
endif

;- Check scaling range for pseudo color images
if (true eq 0) then begin
  if (n_elements(range) eq 0) then begin
    min_value = min(image, max=max_value)
    range = [min_value, max_value]
  endif
  if (n_elements(range) ne 2) then $
    message, 'RANGE keyword must be a 2-element vector'
endif else begin
  if (n_elements(range) gt 0) then $
    message, 'RANGE keyword is not used for True Color images', /continue
endelse

;- Check for supported graphics devices
names = ['WIN', 'MAC', 'X', 'CGM', 'PCL', 'PRINTER', 'PS', 'Z']
result = where((!d.name eq names), count)
if (count eq 0) then message, 'Graphics device is not supported'

;- Get color table information
if ((!d.flags and 256) ne 0) and (!d.window lt 0) then begin
  window, /free, /pixmap
  wdelete, !d.window
endif
if (n_elements(bottom) eq 0) then bottom = 0
if (n_elements(ncolors) eq 0) then ncolors = !d.table_size - bottom

;- Get IDL version number
version = float(!version.release)

;- Check for IDL 5.2 or higher if printer device is selected
if (version lt 5.2) and (!d.name eq 'PRINTER') then $
  message, 'IDL 5.2 or higher is required for PRINTER device support'

;---------------------------------------------------------------------------
;- GET RED, GREEN, AND BLUE COMPONENTS OF TRUE COLOR IMAGE
;---------------------------------------------------------------------------

if (true eq 1) then begin
    case truedim of
      0 : begin
            red = image[0, *, *]
            grn = image[1, *, *]
            blu = image[2, *, *]
      end
      1 : begin
            red = image[*, 0, *]
            grn = image[*, 1, *]
            blu = image[*, 2, *]
      end
      2 : begin
            red = image[*, *, 0]
            grn = image[*, *, 1]
            blu = image[*, *, 2]
      end
  endcase
  red = reform(red, /overwrite)
  grn = reform(grn, /overwrite)
  blu = reform(blu, /overwrite)
endif

;---------------------------------------------------------------------------
;- COMPUTE POSITION FOR IMAGE
;---------------------------------------------------------------------------
```

```
595
596 ;- Save first element of !p.multi
597 multi_first = !p.multi[0]
598
599 ;- Establish image position if not defined
600 if (n_elements(position) eq 0) then begin
601   if (max(!p.multi) eq 0) then begin
602     position = [0.0, 0.0, 1.0, 1.0]
603   endif else begin
604     plot, [0], /nodata, xstyle=4, ystyle=4, xmargin=[0, 0], ymargin=[0, 0]
605     position = [!x.window[0], !y.window[0], !x.window[1], !y.window[1]]
606   endelse
607 endif
608
609 ;- Erase and fill the background if required
610 if (multi_first eq 0) then begin
611   if keyword_set(erase) then erase
612   if (n_elements(background) gt 0) then begin
613     polyfill, [-0.01,  1.01,  1.01, -0.01, -0.01], $
614       [-0.01, -0.01,  1.01,  1.01, -0.01], /normal, color=background[0]
615   endif
616 endif
617
618 ;- Compute image aspect ratio if not defined
619 if (n_elements(aspect) eq 0) then begin
620   case true of
621     0 : result = size(image)
622     1 : result = size(red)
623   endcase
624   dims = result[1:2]
625   aspect = float(dims[1]) / float(dims[0])
626 endif
627
628 ;- Save image xrange and yrange for axis overlays
629 xrange = [0, dims[0]]
630 yrange = [0, dims[1]]
631 if (order eq 1) then yrange = reverse(yrange)
632
633 ;- Set the aspect ratio and margin to fill the position window if requested
634 if keyword_set(usepos) then begin
635   xpos_size = float(!d.x_vsize) * (position[2] - position[0])
636   ypos_size = float(!d.y_vsize) * (position[3] - position[1])
637   aspect_value = ypos_size / xpos_size
638   margin_value = 0.0
639 endif else begin
640   aspect_value = aspect
641   margin_value = margin
642 endelse
643
644 ;- Compute size of displayed image and save output position
645 pos = position
646 case true of
647   0 : imdisp_imsize, image, x0, y0, xsize, ysize, position=pos, $
648         aspect=aspect_value, margin=margin_value
649   1 : imdisp_imsize,   red, x0, y0, xsize, ysize, position=pos, $
650         aspect=aspect_value, margin=margin_value
651 endcase
652 out_pos = pos
653
654 ;-------------------------------------------------------------------------------
655 ;- BYTE-SCALE THE IMAGE IF REQUIRED
656 ;-------------------------------------------------------------------------------
657
658 ;- Choose whether to scale the image or not
659 if (keyword_set(noscale) eq 0) then begin
660
```

```
661   ;- Scale the image
662   case true of
663     0 : scaled = imdisp_imscale(image, bottom=bottom, ncolors=ncolors, $
664           range=range, negative=keyword_set(negative))
665     1 : begin
666           scaled_dims = (size(red))[1:2]
667           scaled = bytarr(scaled_dims[0], scaled_dims[1], 3)
668           scaled[*, *, 0] = imdisp_imscale(red, bottom=0, ncolors=256, $
669             negative=keyword_set(negative))
670           scaled[*, *, 1] = imdisp_imscale(grn, bottom=0, ncolors=256, $
671             negative=keyword_set(negative))
672           scaled[*, *, 2] = imdisp_imscale(blu, bottom=0, ncolors=256, $
673             negative=keyword_set(negative))
674         end
675   endcase
676
677 endif else begin
678
679   ;- Don't scale the image
680   case true of
681     0 : scaled = image
682     1 : begin
683           scaled_dims = (size(red))[1:2]
684           scaled = replicate(red[0], scaled_dims[0], scaled_dims[1], 3)
685           scaled[*, *, 0] = red
686           scaled[*, *, 1] = grn
687           scaled[*, *, 2] = blu
688         end
689   endcase
690
691 endelse
692
693 ;---------------------------------------------------------------------------
694 ;- DISPLAY IMAGE ON PRINTER DEVICE
695 ;---------------------------------------------------------------------------
696
697 if (!d.name eq 'PRINTER') then begin
698
699   ;- Display the image
700   case true of
701     0 : begin
702           device, /index_color
703           tv, scaled, x0, y0, xsize=xsize, ysize=ysize, order=order
704         end
705     1 : begin
706           device, /true_color
707           tv, scaled, x0, y0, xsize=xsize, ysize=ysize, order=order, true=3
708         end
709   endcase
710
711   ;- Draw axes if required
712   if keyword_set(axis) then $
713     plot, [0], /nodata, /noerase, position=out_pos, $
714       xrange=xrange, xstyle=1, yrange=yrange, ystyle=1, $
715       _extra=extra_keywords
716
717   ;- Return to caller
718   return
719
720 endif
721
722 ;---------------------------------------------------------------------------
723 ;- DISPLAY IMAGE ON GRAPHICS DEVICES WHICH HAVE SCALEABLE PIXELS
724 ;---------------------------------------------------------------------------
725
726 if ((!d.flags and 1) ne 0) then begin
```

```
727
728   ;- Display the image
729   case true of
730     0 : tv, scaled, x0, y0, xsize=xsize, ysize=ysize, order=order
731     1 : begin
732           tvlct, r, g, b, /get
733           loadct, 0, /silent
734           tv, scaled, x0, y0, xsize=xsize, ysize=ysize, order=order, true=3
735           tvlct, r, g, b
736         end
737   endcase
738
739   ;- Draw axes if required
740   if keyword_set(axis) then $
741     plot, [0], /nodata, /noerase, position=out_pos, $
742       xrange=xrange, xstyle=1, yrange=yrange, ystyle=1, $
743       _extra=extra_keywords
744
745   ;- Return to caller
746   return
747
748 endif
749
750 ;---------------------------------------------------------------------------
751 ;- RESIZE THE IMAGE
752 ;---------------------------------------------------------------------------
753
754 ;- Resize the image
755 if (keyword_set(noresize) eq 0) then begin
756   if (true eq 0) then begin
757     resized = imdisp_imregrid(scaled, xsize, ysize, interp=keyword_set(interp))
758   endif else begin
759     resized = replicate(scaled[0], xsize, ysize, 3)
760     resized[*, *, 0] = imdisp_imregrid(reform(scaled[*, *, 0]), xsize, ysize, $
761       interp=keyword_set(interp))
762     resized[*, *, 1] = imdisp_imregrid(reform(scaled[*, *, 1]), xsize, ysize, $
763       interp=keyword_set(interp))
764     resized[*, *, 2] = imdisp_imregrid(reform(scaled[*, *, 2]), xsize, ysize, $
765       interp=keyword_set(interp))
766   endelse
767 endif else begin
768   resized = temporary(scaled)
769   x0 = 0
770   y0 = 0
771 endelse
772
773 ;---------------------------------------------------------------------------
774 ;- GET BIT DEPTH FOR THIS DISPLAY
775 ;---------------------------------------------------------------------------
776
777 ;- If this device supports windows, make sure a window has been opened
778 if (!d.flags and 256) ne 0 then begin
779   if (!d.window lt 0) then begin
780     window, /free, /pixmap
781     wdelete, !d.window
782   endif
783 endif
784
785 ;- Set default display depth
786 depth = 8
787
788 ;- Get actual bit depth on supported displays
789 if (!d.name eq 'WIN') or (!d.name eq 'MAC') or (!d.name eq 'X') then begin
790   if (version ge 5.1) then begin
791     device, get_visual_depth=depth
792   endif else begin
```

```
    if (!d.n_colors gt 256) then depth = 24
  endelse
endif

;-------------------------------------------------------------------------------
;- SELECT DECOMPOSED COLOR MODE (ON OR OFF) FOR 24-BIT DISPLAYS
;-------------------------------------------------------------------------------

if (!d.name eq 'WIN') or (!d.name eq 'MAC') or (!d.name eq 'X') then begin
  if (depth gt 8) then begin
    if (version ge 5.2) then device, get_decomposed=entry_decomposed else $
      entry_decomposed = 0
    if (true eq 1) or (channel gt 0) then device, decomposed=1 else $
      device, decomposed=0
  endif
endif

;-------------------------------------------------------------------------------
;- DISPLAY THE IMAGE
;-------------------------------------------------------------------------------

;- If the display is 8-bit and the image is true color,
;- convert image from true color to indexed color
if (depth le 8) and (true eq 1) then begin
  resized = color_quan(temporary(resized), 3, r, g, b, $
    colors=ncolors, dither=keyword_set(dither)) + byte(bottom)
  tvlct, r, g, b, bottom
  true = 0
endif

;- Set channel value for supported devices
if (!d.name eq 'WIN') or (!d.name eq 'MAC') or (!d.name eq 'X') then begin
  channel_value = channel
endif else begin
  channel_value = 0
endelse

;- Display the image
case true of
  0 : tv, resized, x0, y0, order=order, channel=channel_value
  1 : tv, resized, x0, y0, order=order, true=3
endcase

;-------------------------------------------------------------------------------
;- RESTORE THE DECOMPOSED COLOR MODE FOR 24-BIT DISPLAYS
;-------------------------------------------------------------------------------

if (!d.name eq 'WIN') or (!d.name eq 'MAC') or (!d.name eq 'X') then begin
  if (depth gt 8) then device, decomposed=entry_decomposed
endif

;-------------------------------------------------------------------------------
;- DRAW AXES IF REQUIRED
;-------------------------------------------------------------------------------

if keyword_set(axis) then $
  plot, [0], /nodata, /noerase, position=out_pos, $
    xrange=xrange, xstyle=1, yrange=yrange, ystyle=1, $
    _extra=extra_keywords

END

;;;;;###############################################################################
;###############################################################################
;+
; NAME:
```

```
;       GETCOLOR
;
; PURPOSE:
;       The original purpose of this function was to enable the
;       user to specify one of the 16 colors supported by the
;       McIDAS color map by name. Over time, however, the function
;       has become a general purpose function for handling and
;       supporting drawing colors in a device-independent way.
;       In particular, I have been looking for ways to write color
;       handling code that will work transparently on both 8-bit and
;       24-bit machines. On 24-bit machines, the code should work the
;       same where color decomposition is turned on or off. The program
;       now supports 88 colors.
;
; AUTHOR:
;       FANNING SOFTWARE CONSULTING:
;       David Fanning, Ph.D.
;       1645 Sheely Drive
;       Fort Collins, CO 80526 USA
;       Phone: 970-221-0438
;       E-mail: davidf@dfanning.com
;       Coyote's Guide to IDL Programming: http://www.dfanning.com
;
; CATEGORY:
;       Graphics, Color Specification.
;
; CALLING SEQUENCE:
;       result = GETCOLOR(color, index)
;
; OPTIONAL INPUT PARAMETERS:
;       COLOR: A string with the "name" of the color. Valid names are:
;           black
;           magenta
;           cyan
;           yellow
;           green
;           red
;           blue
;           navy
;           pink
;           aqua
;           orchid
;           sky
;           beige
;           charcoal
;           gray
;           white
;
;           The color YELLOW is returned if the color name can't be resolved.
;           Case is unimportant.
;
;           If the function is called with just this single input parameter,
;           the return value is either a 1-by-3 array containing the RGB values of
;           that particular color, or a 24-bit integer that can be "decomposed" into
;           that particular color, depending upon the state of the TRUE keyword and
;           upon whether color decomposition is turned on or off. The state of color
;           decomposition can ONLY be determined if the program is being run in
;           IDL 5.2 or higher.
;
;       INDEX: The color table index where the specified color should be loaded.
;           If this parameter is passed, then the return value of the function is the
;           index number and not the color triple. (If color decomposition is turned
;           on AND the user specifies an index parameter, the color is loaded in the
;           color table at the proper index, but a 24-bit value is returned to the
;           user in IDL 5.2 and higher. This assumes the INDEXED keyword is NOT set.)
;
```

```
;       If no positional parameter is present, then the return value is either a 16-by-3
;       byte array containing the RGB values of all 16 colors or it is a 16-element
;       long integer array containing color values that can be decomposed into colors.
;       The 16-by-3 array is appropriate for loading color tables with the TVLCT command:
;
;           Device, Decomposed=0
;           colors = GetColor()
;           TVLCT, colors, 100
;
;
; INPUT KEYWORD PARAMETERS:
;
;       NAMES: If this keyword is set, the return value of the function is
;              a 88-element string array containing the names of the colors.
;              These names would be appropriate, for example, in building
;              a list widget with the names of the colors. If the NAMES
;              keyword is set, the COLOR and INDEX parameters are ignored.
;
;                 listID = Widget_List(baseID, Value=GetColor(/Names), YSize=16)
;
;       INDEXED:  If this keyword is set, the return value is always an index
;              into the color table. In the absence of a color table INDEX
;              parameter, the color is loaded at !P.COLOR < (!D.Table_Size-1).
;
;       LOAD:  If this keyword is set, all 88 colors are automatically loaded
;              starting at the color index specified by the START keyword.
;              Note that setting this keyword means that the return value of the
;              function will be a structure, with each field of the structure
;              corresponding to a color name. The value of each field will be
;              an index number (set by the START keyword) corresponding to the
;              associated color, or a 24-bit long integer value that creates the
;              color on a true-color device. What you have as the field values is
;              determined by the TRUE keyword or whether color decomposition is on
;              or off in the absense of the TRUE keyword. It will either be a 1-by-3
;              byte array or a long integer value.
;
;       START: The starting color index number if the LOAD keyword is set. This keyword
;              value is ignored unless the LOAD keyword is also set. The keyword is also
;              ignored if the TRUE keyword is set or if color decomposition in on in
;              IDL 5.2 and higher. The default value for the START keyword is
;              !D.TABLE_SIZE - 89.
;
;       TRUE:  If this keyword is set, the specified color triple is returned
;              as a 24-bit integer equivalent. The lowest 8 bits correspond to
;              the red value; the middle 8 bits to the green value; and the
;              highest 8 bits correspond to the blue value. In IDL 5.2 and higher,
;              if color decomposition is turned on, it is as though this keyword
;              were set.
;
; COMMON BLOCKS:
;       None.
;
; SIDE EFFECTS:
;       None.
;
; RESTRICTIONS:
;       The TRUE keyword causes the START keyword to be ignored.
;       The NAMES keyword causes the COLOR, INDEX, START, and TRUE parameters to be ignored.
;       The COLOR parameter is ignored if the LOAD keyword is used.
;       On systems where it is possible to tell the state of color decomposition
;       (i.e., IDL 5.2 and higher), a 24-bit value (or values) is automatically
;       returned if color decomposition is ON.
;
; EXAMPLE:
;       To load a yellow color in color index 100 and plot in yellow, type:
;
```

```
;           yellow = GETCOLOR('yellow', 100)
;           PLOT, data, COLOR=yellow
;
;       or,
;
;           PLOT, data, COLOR=GETCOLOR('yellow', 100)
;
;       To do the same thing on a 24-bit color system with decomposed color on, type:
;
;           PLOT, data, COLOR=GETCOLOR('yellow', /TRUE)
;
;       or in IDL 5.2 and higher,
;
;           DEVICE, Decomposed=1
;           PLOT, data, COLOR=GETCOLOR('yellow')
;
;       To load all 88 colors into the current color table, starting at
;       color index 100, type:
;
;           TVLCT, GETCOLOR(), 100
;
;       To add the color names to a list widget:
;
;            listID = Widget_List(baseID, Value=GetColor(/Names), YSize=16)
;
;       To load all 88 colors and have the color indices returned in a structure:
;
;            DEVICE, Decomposed=0
;            colors = GetColor(/Load, Start=1)
;            HELP, colors, /Structure
;            PLOT, data, COLOR=colors.yellow
;
;       To get the direct color values as 24-bit integers in color structure fields:
;
;            DEVICE, Decomposed=1
;            colors = GetColor(/Load)
;            PLOT, data, COLOR=colors.yellow
;
;       Note that the START keyword value is ignored if on a 24-bit device,
;       so it is possible to write completely device-independent code by
;       writing code like this:
;
;            colors = GetColor(/Load)
;            PLOT, data, Color=colors.yellow
;
; MODIFICATION HISTORY:
;       Written by: David Fanning, 10 February 96.
;       Fixed a bug in which N_ELEMENTS was spelled wrong. 7 Dec 96. DWF
;       Added the McIDAS colors to the program. 24 Feb 99. DWF
;       Added the INDEX parameter to the program 8 Mar 99. DWF
;       Added the NAMES keyword at insistence of Martin Schultz. 10 Mar 99. DWF
;       Reorderd the colors so black is first and white is last. 7 June 99. DWF
;       Added automatic recognition of DECOMPOSED=1 state. 7 June 99. DWF
;       Added LOAD AND START keywords. 7 June 99. DWF.
;       Replaced GOLD with CHARCOAL color. 28 Oct 99. DWF.
;       Added INDEXED keyword to force indexed color mode. 28 Oct 99. DWF.
;       Fixed problem of "aqua" and "pink" being mixed up. 18 Mar 00. DWF.
;       Changed ON_ERROR from 1 to 2, and improved error handling. 2 Aug 00. DWF.
;       Increased the known colors from 16 to 88. 19 October 2000. DWF.
;       Fixed typos in which "thisColor" was written as "theColor". 10 AUG 2001. DWF.
;-
;
;#############################################################################
;
; LICENSE
;
```

```
; This software is OSI Certified Open Source Software.
; OSI Certified is a certification mark of the Open Source Initiative.
;
; Copyright \ufffd 2000 Fanning Software Consulting.
;
; This software is provided "as-is", without any express or
; implied warranty. In no event will the authors be held liable
; for any damages arising from the use of this software.
;
; Permission is granted to anyone to use this software for any
; purpose, including commercial applications, and to alter it and
; redistribute it freely, subject to the following restrictions:
;
; 1. The origin of this software must not be misrepresented; you must
;    not claim you wrote the original software. If you use this software
;    in a product, an acknowledgment in the product documentation
;    would be appreciated, but is not required.
;
; 2. Altered source versions must be plainly marked as such, and must
;    not be misrepresented as being the original software.
;
; 3. This notice may not be removed or altered from any source distribution.
;
; For more information on Open Source Software, visit the Open Source
; web site: http://www.opensource.org.
;
;#############################################################################


FUNCTION COLOR24, number

   ; This FUNCTION accepts a [red, green, blue] triple that
   ; describes a particular color and returns a 24-bit long
   ; integer that is equivalent to that color. The color is
   ; described in terms of a hexidecimal number (e.g., FF206A)
   ; where the left two digits represent the blue color, the
   ; middle two digits represent the green color, and the right
   ; two digits represent the red color.
   ;
   ; The triple can be either a row or column vector of 3 elements.

ON_ERROR, 1

IF N_ELEMENTS(number) NE 3 THEN $
   MESSAGE, 'Augument must be a three-element vector.'

IF MAX(number) GT 255 OR MIN(number) LT 0 THEN $
   MESSAGE, 'Argument values must be in range of 0-255'

base16 = [[1L, 16L], [256L, 4096L], [65536L, 1048576L]]

num24bit = 0L

FOR j=0,2 DO num24bit = num24bit + ((number(j) MOD 16) * base16(0,j)) + $
   (Fix(number(j)/16) * base16(1,j))

RETURN, num24bit
END ; ***********************  of COLOR24  *****************************


FUNCTION GETCOLOR, thisColor, index, TRUE=truecolor, $
   NAMES=colornames, LOAD=load, START=start, INDEXED=indexedcolor

   ; Set up the color vectors.

```

```
   names =  ['White']
   rvalue = [ 255]
   gvalue = [ 255]
   bvalue = [ 255]
   names  = [ names,        'Snow',     'Ivory','Light Yellow',   'Cornsilk',      'Beige',   'Seashell'
   rvalue = [ rvalue,          255,          255,          255,          255,          245,          255
   gvalue = [ gvalue,          250,          255,          255,          248,          245,          245
   bvalue = [ bvalue,          250,          240,          224,          220,          220,          238
   names  = [ names,       'Linen','Antique White',    'Papaya',     'Almond',     'Bisque',  'Moccasin'
   rvalue = [ rvalue,          250,          250,          255,          255,          255,          255
   gvalue = [ gvalue,          240,          235,          239,          235,          228,          228
   bvalue = [ bvalue,          230,          215,          213,          205,          196,          181
   names  = [ names,       'Wheat',  'Burlywood',        'Tan', 'Light Gray',   'Lavender','Medium Gray'
   rvalue = [ rvalue,          245,          222,          210,          230,          230,          210
   gvalue = [ gvalue,          222,          184,          180,          230,          230,          210
   bvalue = [ bvalue,          179,          135,          140,          230,          250,          210
   names  = [ names,        'Gray', 'Slate Gray',  'Dark Gray',   'Charcoal',      'Black', 'Light Cyan'
   rvalue = [ rvalue,          190,          112,          110,           70,            0,          224
   gvalue = [ gvalue,          190,          128,          110,           70,            0,          255
   bvalue = [ bvalue,          190,          144,          110,           70,            0,          255
   names  = [ names, 'Powder Blue',   'Sky Blue', 'Steel Blue','Dodger Blue', 'Royal Blue',       'Blue'
   rvalue = [ rvalue,          176,          135,           70,           30,           65,            0
   gvalue = [ gvalue,          224,          206,          130,          144,          105,            0
   bvalue = [ bvalue,          230,          235,          180,          255,          225,          255
   names  = [ names,        'Navy',   'Honeydew', 'Pale Green','Aquamarine','Spring Green',       'Cyan'
   rvalue = [ rvalue,            0,          240,          152,          127,            0,            0
   gvalue = [ gvalue,            0,          255,          251,          255,          250,          255
   bvalue = [ bvalue,          128,          240,          152,          212,          154,          255
   names  = [ names,   'Turquoise', 'Sea Green','Forest Green','Green Yellow','Chartreuse', 'Lawn Green'
   rvalue = [ rvalue,           64,           46,           34,          173,          127,          124
   gvalue = [ gvalue,          224,          139,          139,          255,          255,          252
   bvalue = [ bvalue,          208,           87,           34,           47,            0,            0
   names  = [ names,       'Green', 'Lime Green', 'Olive Drab',     'Olive','Dark Green','Pale Goldenrod'
   rvalue = [ rvalue,            0,           50,          107,           85,            0,          238
   gvalue = [ gvalue,          255,          205,          142,          107,          100,          232
   bvalue = [ bvalue,            0,           50,           35,           47,            0,          170
   names  = [ names,       'Khaki', 'Dark Khaki',     'Yellow',       'Gold','Goldenrod','Dark Goldenrod'
   rvalue = [ rvalue,          240,          189,          255,          255,          218,          184
   gvalue = [ gvalue,          230,          183,          255,          215,          165,          134
   bvalue = [ bvalue,          140,          107,            0,            0,           32,           11
   names  = [ names,'Saddle Brown',       'Rose',       'Pink', 'Rosy Brown','Sandy Brown',       'Peru'
   rvalue = [ rvalue,          139,          255,          255,          188,          244,          205
   gvalue = [ gvalue,           69,          228,          192,          143,          164,          133
   bvalue = [ bvalue,           19,          225,          203,          143,           96,           63
   names  = [ names,  'Indian Red',  'Chocolate',     'Sienna','Dark Salmon',    'Salmon','Light Salmon'
   rvalue = [ rvalue,          205,          210,          160,          233,          250,          255
   gvalue = [ gvalue,           92,          105,           82,          150,          128,          160
   bvalue = [ bvalue,           92,           30,           45,          122,          114,          122
   names  = [ names,      'Orange',      'Coral', 'Light Coral',  'Firebrick',      'Brown',  'Hot Pink'
   rvalue = [ rvalue,          255,          255,          240,          178,          165,          255
   gvalue = [ gvalue,          165,          127,          128,           34,           42,          105
   bvalue = [ bvalue,            0,           80,          128,           34,           42,          180
   names  = [ names,   'Deep Pink',    'Magenta',     'Tomato', 'Orange Red',        'Red', 'Violet Red'
   rvalue = [ rvalue,          255,          255,          255,          255,          255,          208
   gvalue = [ gvalue,           20,            0,           99,           69,            0,           32
   bvalue = [ bvalue,          147,          255,           71,            0,            0,          144
   names  = [ names,      'Maroon',    'Thistle',       'Plum',     'Violet',    'Orchid','Medium Orchid'
   rvalue = [ rvalue,          176,          216,          221,          238,          218,          186
   gvalue = [ gvalue,           48,          191,          160,          130,          112,           85
   bvalue = [ bvalue,           96,          216,          221,          238,          214,          211
   names  = [ names, 'Dark Orchid','Blue Violet',     'Purple' ]
   rvalue = [ rvalue,          153,          138,          160 ]
   gvalue = [ gvalue,           50,           43,           32 ]
   bvalue = [ bvalue,          204,          226,          240 ]

   ; Did the user ask for a specific color? If not, return
```

```
   ; all the colors. If the user asked for a specific color,
   ; find out if a 24-bit value is required. Return to caller
   ; if an error occurs.

ON_Error, 2
ncolors = N_Elements(names)

np = N_Params()
IF N_Elements(start) EQ 0 THEN start = !D.TABLE_SIZE - ncolors - 1 ELSE start = start < (!D.TABLE_SIZE -

   ; User ask for the color names?

IF Keyword_Set(colornames) THEN RETURN, Reform(names, 1, N_Elements(names)) $
    ELSE names = StrUpCase(StrCompress(StrTrim(names,2), /Remove_All))

   ; If no positional parameter, return all colors.

IF np EQ 0 THEN BEGIN

   ; Did the user want a 24-bit value? If so, call COLOR24.

   IF Keyword_Set(trueColor) THEN BEGIN
      returnColor = LonArr(ncolors)
      FOR j=0,ncolors-1 DO returnColor[j] = Color24([rvalue[j], gvalue[j], bvalue[j]])

         ; If LOAD keyword set, return a color structure.

      IF Keyword_Set(load) THEN BEGIN
         returnValue = Create_Struct('white', returnColor[0])
         FOR j=1,ncolors-1 DO returnValue = Create_Struct(returnValue, names[j], returnColor[j])
         returnColor = returnValue
      ENDIF

      RETURN, returnColor
   ENDIF

   ; If color decomposition is ON and INDEXED is not set, return 24-bit values.

   IF Float(!Version.Release) GE 5.2 THEN BEGIN
      IF (!D.Name EQ 'X' OR !D.Name EQ 'WIN' OR !D.Name EQ 'MAC') THEN BEGIN
         Device, Get_Decomposed=decomposedState
      ENDIF ELSE decomposedState = 0
      IF Keyword_Set(indexedcolor) THEN decomposedState = 0
      IF decomposedState EQ 1 THEN BEGIN
         returnColor = LonArr(ncolors)
         FOR j=0,ncolors-1 DO returnColor[j] = Color24([rvalue[j], gvalue[j], bvalue[j]])
         IF Keyword_Set(load) THEN BEGIN
            returnValue = Create_Struct('white', returnColor[0])
            FOR j=1,ncolors-1 DO returnValue = Create_Struct(returnValue, names[j], returnColor[j])
            RETURN, returnValue
         ENDIF
         RETURN, returnColor
      ENDIF

      IF Keyword_Set(load) THEN BEGIN
         TVLCT, Reform([rvalue, gvalue, bvalue], ncolors, 3), start
         returnValue = Create_Struct('white', start)
         FOR j=1,ncolors-1 DO returnValue = Create_Struct(returnValue, names[j], start+j)
         RETURN, returnValue
      ENDIF

      returnColor = REFORM([rvalue, gvalue, bvalue], ncolors, 3)
      RETURN, returnColor

   ENDIF

```

```
   IF Keyword_Set(load) THEN BEGIN
      TVLCT, Reform([rvalue, gvalue, bvalue], ncolors, 3), start
      returnValue = Create_Struct('white', start)
      FOR j=1,ncolors-1 DO returnValue = Create_Struct(returnValue, names[j], start+j)
      RETURN, returnValue
   ENDIF

   returnColor = REFORM([rvalue, gvalue, bvalue], ncolors, 3)
   RETURN, returnColor

ENDIF

   ; Make sure the color parameter is an uppercase string.

varInfo = SIZE(thisColor)
IF varInfo(varInfo(0) + 1) NE 7 THEN $
   MESSAGE, 'The color name must be a string.'
thisColor = StrUpCase(StrCompress(StrTrim(thisColor,2), /Remove_All))

   ; Check synonyms of colors.

IF StrUpCase(thisColor) EQ 'GREY' THEN thisColor = 'GRAY'
IF StrUpCase(thisColor) EQ 'AQUA' THEN thisColor = 'AQUAMARINE'
IF StrUpCase(thisColor) EQ 'SKYBLUE' THEN thisColor = 'SKY BLUE'
IF StrUpCase(thisColor) EQ 'LIGHTGREY' THEN thisColor = 'LIGHTGRAY'
IF StrUpCase(thisColor) EQ 'MEDIUMGREY' THEN thisColor = 'MEDIUMGRAY'
IF StrUpCase(thisColor) EQ 'SLATEGREY' THEN thisColor = 'SLATEGRAY'
IF StrUpCase(thisColor) EQ 'DARKGREY' THEN thisColor = 'DARKGRAY'
IF StrUpCase(thisColor) EQ 'SKY' THEN thisColor = 'SKY BLUE'
IF StrUpCase(thisColor) EQ 'NAVY BLUE' THEN thisColor = 'NAVY'
IF StrUpCase(thisColor) EQ 'NAVYBLUE' THEN thisColor = 'NAVY'


   ; Get the color triple for this color.

colorIndex = WHERE(names EQ thisColor)

   ; If you can't find it. Issue an informational message,
   ; set the index to a YELLOW color, and continue.

IF colorIndex(0) LT 0 THEN BEGIN
   MESSAGE, "Can't find color " + thisColor + ". Returning " + StrUpCase(names[0]) + ".", /INFORMATIONAL
   thisColor = names[0]
   colorIndex = 0
ENDIF

   ; Get the color triple.

r = rvalue(colorIndex)
g = gvalue(colorIndex)
b = bvalue(colorIndex)
returnColor = REFORM([r, g, b], 1, 3)

   ; Did the user want a 24-bit value? If so, call COLOR24.

IF KEYWORD_SET(trueColor) THEN BEGIN
   returnColor = COLOR24(returnColor)
   RETURN, returnColor[0]
ENDIF

   ; If color decomposition is ON and INDEXED is OFF,, return 24-bit value.

IF Float(!Version.Release) GE 5.2 THEN BEGIN

   IF (!D.Name EQ 'X' OR !D.Name EQ 'WIN' OR !D.Name EQ 'MAC') THEN BEGIN
      Device, Get_Decomposed=decomposedState
```

```
   ENDIF ELSE decomposedState = 0
   IF Keyword_Set(indexedcolor) THEN decomposedState = 0

   IF decomposedState EQ 1 THEN BEGIN

         ; Before you change return color, load index if requested.

      IF N_Elements(index) NE 0 THEN BEGIN
         index = 0 > index < (!D.Table_Size-1)
         TVLCT, returnColor, index
      ENDIF

      returnColor = COLOR24(returnColor)
      RETURN, returnColor[0]
   ENDIF
ENDIF

   ; Did the user specify a color index? If so, load it.

IF N_Elements(index) NE 0 THEN BEGIN
   index = 0 > index < (!D.Table_Size-1)
   TVLCT, returnColor, index
   returnColor = index
   RETURN, returnColor[0]
ENDIF

   ; Did the user specify INDEXED color? If so, load it.

IF Keyword_Set(indexedColor) THEN BEGIN
   TVLCT, returnColor, !P.Color
   returnColor = !P.Color < (!D.Table_Size -1)
   RETURN, returnColor[0]
ENDIF

RETURN, returnColor
END

;;;######################
;;;######################
;;;######################
;;;######################
;;;######################
pro calstar, myrun
nrun=myrun-1

outrun='master_run'+strtrim(myrun,2)+'.tab'  ;file created
print, outrun

readcol, "rawtables.tab", log, dirin, f='(a,a)'  ;;local logs

;make master table
readcol_loc, "master.tab",  OBJECTref, RAref, DECref, flag1, nobs1, frame1, Nframeref, ExpTref,cutref,cut
flag2, nobs2, frame2, file2,  Nframe2, ExpT2, zeropointref2, zeropoint2, mag2, err2,$
flag3, nobs3, frame3, file3,  Nframe3, ExpT3, zeropointref3, zeropoint3, mag3, err3,$
flag4, nobs4, frame4, file4,  Nframe4, ExpT4, zeropointref4, zeropoint4, mag4, err4,$
flag5, nobs5, frame5, file5,  Nframe5, ExpT5, zeropointref5, zeropoint5, mag5, err5,$
flag6, nobs6, frame6, file6,  Nframe6, ExpT6, zeropointref6, zeropoint6, mag6, err6,$
flag7, nobs7, frame7, file7,  Nframe7, ExpT7, zeropointref7, zeropoint7, mag7, err7,$
flag8, nobs8, frame8, file8,  Nframe8, ExpT8, zeropointref8, zeropoint8, mag8, err8,$
flag9, nobs9, frame9, file9,  Nframe9, ExpT9, zeropointref9, zeropoint9, mag9, err9,$
flag10, nobs10, frame10, file10 , Nframe10, ExpT10, zeropointref10, zeropoint10, mag10, err10,$
flag11, nobs11, frame11, file11 , Nframe11, ExpT11, zeropointref11, zeropoint11, mag11, err11,$
format='(a,d,d,i,i,a,i,f,f,f,f,f,(i,i,a,a,i,f,f,f,f,f),(i,i,a,a,i,f,f,f,f,f),(i,i,a,a,i,f,f,f,f,f),(i,i,a

;;########################################
;OBJECTref, RAref, DECref, flag1, nobs1, frame1,         Nframeref, ExpTref,cutref,cutmaxref,magref, errr
```

```
;                          flag2, nobs2, frame2, file2,  Nframe2,   ExpT2, zeropointref2, zeropoint2, mag

;;#######################################
;ok number of observations per run
nobs=intarr(11,n_elements(raref))  ;;;; obs rep in each log
nobs(0,*)=nobs1
nobs(1,*)=nobs2
nobs(2,*)=nobs3
nobs(3,*)=nobs4
nobs(4,*)=nobs5
nobs(5,*)=nobs6
nobs(6,*)=nobs7
nobs(7,*)=nobs8
nobs(8,*)=nobs9
nobs(9,*)=nobs10
nobs(10,*)=nobs11
;;#######################################

;ok number of observations per run
flagif=intarr(11,n_elements(raref))  ;;;;; flag=flag not used,
flagif(0,*)=flag1
flagif(1,*)=flag2
flagif(2,*)=flag3
flagif(3,*)=flag4
flagif(4,*)=flag5
flagif(5,*)=flag6
flagif(6,*)=flag7
flagif(7,*)=flag8
flagif(8,*)=flag9
flagif(9,*)=flag10
flagif(10,*)=flag11


;;;#############################################################################################

get_lun, lun2
openw, lun2, "pippo2"
get_lun, lun10
openw, lun10, "masterflag.tab"


for j=0,n_elements(raref)-1 do begin
filecal=OBJECTref[j]+strtrim(string(raref[j],f='(d10.6)'),2)+'.1'+strtrim(1,2)+'_'+'0'+".tabcal.stars"

printf, lun10, OBJECTref[j],RAref[j], DECref[j],filecal,1, f='( a30,x,d10.6,x,d10.6,x,a39,x,i1)'

if(OBJECTref[j] ne 'none' and j le 56) then begin
namexx=OBJECTref[j]
print,  OBJECTref[j],RAref[j], DECref[j],filecal,1, f='( a30,x,d10.6,x,d10.6,x,a39,x,i1)'

cutcal=13.
if (namexx eq "star05ij") then cutcal=11.5d0
if (namexx eq "star06ij") then cutcal=11.5d0
if (namexx eq "star09ij") then cutcal=11.5d0
if (namexx eq "star21ij") then cutcal=10.5d0
if (namexx eq "star33ij") then cutcal=10.5d0
pp=where(OBJECTref eq namexx) ;;index of this target in the master.tab
rep=nobs(nrun,[pp[0]])        ;;repetition counter

;if(magref[j] gt 0 or mag2[j] gt 0 or mag3[j] gt 0 or mag4[j] gt 0 or mag5[j] gt 0 or mag6[j] gt 0 or mag
if( flagif(nrun,[pp[0]]) eq 1 ) then begin
print, namexx,rep
help, OBJECTref, rep

;-----------------------------------------------------------------
mylog=log[nrun]        ;local log of the selected run
```

```
mydirin=dirin[nrun]


for pk=0, rep[0]-1 do begin
resid1=0
resid2=0
file2[pp]='none'
fileout=namexx+strtrim(string(raref[pp],f='(d10.6)'),2)+'.1'+strtrim(myrun,2)+'_'+strtrim(pk,2)+".tab"
fileout="tmp/"+fileout[0]
framex='none'
Nframex=-1
ExpTx=0.0
magx=0.0
errx=0.0
zeropointx=0.0
julien=0.0
zepref=0.0
zpreferr=0.0
zerop2=0.0
zpoint2err=0.0
flagx = 0

calibrel, mylog,pk, Raref[pp], Decref[pp], cutcal, zepref=zepref, zpreferr=zpreferr, zerop2=zerop2, zpoin
mydirin,fileout, flagx=flagx, framex=framex,  Nframex=Nframex, ExpTx=ExpTx, julien=julien,namezpx=namezpx
flagrefF, nobsrefF, framerefF, NframerefF, ExpTrefF,cutrefF,cutmaxrefF,magrefF, errrefF


zeropointrefx=zepref[0]
zeropointx=zerop2[0]

if (flagx eq 1 ) then begin
  if nrun ne 0 then recal, mydirin, fileout,resid1=resid1,resid2=resid2, cutcal, cutmaxref[pp], a,b,c,d
  ;;;;;if nrun eq 0 then recalzero, mydirin, fileout,resid1=resid1,resid2=resid2, cutcal, cutmaxref[pp],
  if nrun eq 0 then recal, mydirin, fileout,resid1=resid1,resid2=resid2, cutcal, cutmaxref[pp], a,b,c,d
  magref[pp]=a ;+resid1[0]
  errref[pp]=b
  magx=c       ;+resid2[0]
  errx=d
  file2[pp]=fileout
  zeropointrefx=zepref[0]  ;+resid1[0]
  zeropointx=zerop2[0]     ;+resid2[0]
endif

print, "The recal submacro only plots. The zero offsets are based on 2MASS and untouched.If you loop for
;;;###########################################################################################

printf, lun2 , OBJECTref[j], RAref[j], DECref[j], flagrefF, nobsrefF, framerefF, NframerefF, ExpTrefF,cut
flagx, rep, framex, file2[j],  Nframex, ExpTx, zeropointrefx,zpreferr,resid1[0], zeropointx, zpoint2err,r
format='(a14,x,d10.6,x,d10.6,x,i3,x,i3,x,a25,x,i3,x,f5.2,x,f5.2,x,f5.2,x,f6.3,x,f6.3,1(x,i3,x,i3,x,a25,x,


endfor

endif
endif
endfor
close, lun2
free_lun, lun2
close, lun10
free_lun, lun10

cmd="sort -k 15"+" pippo2 >"+outrun
spawn, cmd

end

```

```
;;#############################################################


pro   calibrel,  input, nrep , raxx, decxx, cutcal, $
zepref=zepref, zpreferr=zpreferr, zerop2=zerop2, zpoint2err=zpoint2err, $
dirin, fileout,  flagx=flagx, framex=framex,  Nframex=Nframex, ExpTx=ExpTx, julien=julien, namezpx=namezp
flagrefF, nobsrefF, framerefF, NframerefF, ExpTrefF,cutrefF,cutmaxrefF,magrefF, errrefF


;;;;;;#####################################################
;;;;reference catalog (epoch 1) ...retrieving the corresponding reference target in the log.ref
readcol, "log.ref.tab",  OBJECTref, RAref, DECref, Nframeref, $   ;log of run 1 ---taken as reference
ExpTref, DATEref, Framestartref, Framendref, juldayref,secz,cutref,cutmaxref, dirrefa, $
format='(a,d,d,l,f,a,a,a,a,f,f,f,a)'

print, DECxx,RAXX
factor=cos( DECxx[0]/180.*!pi)
distanza=sqrt(((RAref-raxx[0])*factor[0])^2+(DECref-decxx[0])^2)
pp=where(distanza eq min(distanza) and distanza*60.0d0 lt 2.0,cc)
if(cc gt 1 ) then print, OBJECTref[pp], "   not sure"
if(cc gt 1 ) then print, "GOT more than one match, OPS"
if(cc gt 1 ) then stop
print, distanza*60.
print, raxx, decxx, " cavallo"

;;;;;;#####################################################
;;Build the name of the observation used as a reference and the corresponding calibrated photometry.

  la=pp  ;;;how to select here coordinates
  dirref=dirrefa[la]   ;"../REDUCED/run-16B/IRframesRED/"
  proot=strpos(framestartref[la],'.')
  root=strmid(framestartref[la],0,proot+1)
  nn=strmid(framestartref[la],proot+1,4)
  newind=float(nn+0)
  fileref=dirref+root+string(newind[0],f='(f05.0)')+'ave2.tab'  ;idetified final catalog of that field in
  print, fileref

;;;;;;#####################################################

;;local log
readcol, input,  OBJECT, RA, DEC, Nframe, ExpT, DATE, Framestart,$  ;;;second log
Framend, julday,secz,cut,cutmax,fwhm, r1,r2,s1,s2,hampx,namezp,magzp,format='(a,d,d,l,f,a,a,a,a,f,f,f,(f,

      x1=raref[pp[0]]
      y1=decref[pp[0]]
      crossid, x1, y1, ra, dec, indfound, cc

print, indfound,nrep
help, indfound, nrep
print, OBJECT[indfound]
indref=[indfound[nrep]]
dirout=dirin
proot=strpos(framestart[indref],'.')
root=strmid(framestart[indref],0,proot+1)
nn=strmid(framestart[indref],proot+1,4)
newind=float(nn+0)
file=dirout+root+string(newind[0],f='(f05.0)')+'ave2.tab'
;idetified final catalog of that field in run xx

flagx  =hampx[indref]
framex =Framestart[indref]
Nframex=Nframe[indref]
ExpTx  =ExpT[indref]
julien =julday[indref]
namezpx=namezp[indref]
magzpx =magzp[indref]
```

```
1585
1586 if( flagx ne 1 ) then return
1587 print, nrep , "  llll"
1588
1589 ;;;;;;#####################################################
1590
1591   print, fileref[0], " RUN 1"
1592   print, file[0],    " RUN 2"
1593   ;fileref="../REDUCED/run-16B/IRframesRED/binir160809.0188.ave2.tab"
1594   ;file=   "../REDUCED/run-19A/IRframesRED/binir190417.0539.ave2.tab"
1595   readcol,fileref[0], idref, xfindref,yfindref,magref,magerrref, skyref,skyerrref, $
1596   nframeref, xstarref,ystarref,starnameref, rafindref,decfindref,raseltref,decseltref,$
1597   jmagseltref,jerrseltref,zpref,zperrref,$
1598   format='(i,f,f,f,f,f,f,i,f,f,a,d,d,d,d,f,f,f,f)'
1599
1600   print, file[0]
1601   readcol,file[0], id, xfind,yfind,mag,magerr, sky,skyerr, $
1602   nframe, xstar,ystar,starname, rafind,decfind,raselt,decselt,jmagselt,jerrselt,zp,zperr,$
1603   format='(i,f,f,f,f,f,f,i,f,f,a,d,d,d,d,f,f,f,f)'
1604        r_delta=3.d0/3600.0d0
1605       x1=rafind
1606       y1=decfind
1607       maglo=x1*0+9
1608       magerrlo=x1*0+9
1609       magreflo=rafindref*0+9
1610       magerrreflo=rafindref*0+9
1611       ;;cc_starcatalog,rafindref,decfindref,magreflo,magerrreflo,x1,y1,maglo,magerrlo,r_delta,matchid,mat
1612       ;;cc_starcatalog22,rafindref,decfindref,magreflo,magerrreflo,x1,y1,maglo,magerrlo,r_delta,matchid,m
1613       cc_starcatalog3_mon,rafindref,decfindref,magreflo,magerrreflo,x1,y1,maglo,magerrlo,r_delta,matchid,
1614       indfound=where(matchid ge 0, countglimpse)
1615
1616
1617       rarun2    =dblarr(n_elements(idref))+99.99d0
1618       decrun2   =dblarr(n_elements(idref))+99.99d0
1619       magrun2   =dblarr(n_elements(idref))+99.99d0
1620       magerrrun2=dblarr(n_elements(idref))+99.99d0
1621       skyrun2   =dblarr(n_elements(idref))+99.99d0
1622       skyerrrun2=dblarr(n_elements(idref))+99.99d0
1623
1624
1625       rarun2[indfound]    =rafind[matchid[indfound]]
1626       decrun2[indfound]   =decfind[matchid[indfound]]
1627       magrun2[indfound]   =mag[matchid[indfound]]+zp[matchid[indfound]]
1628       magerrrun2[indfound]=magerr[matchid[indfound]]
1629       skyrun2[indfound]   =sky[matchid[indfound]]
1630       skyerrrun2[indfound]=skyerr[matchid[indfound]]
1631
1632       magref=magref+zpref
1633 ;for j=24,24 do print, idref[j], xfindref[j],yfindref[j], magref[j],magrun2[j],rarun2[j]
1634 ;stop
1635
1636 get_lun, lun
1637 openw, lun, fileout
1638
1639 for j=0,n_elements(idref)-1 do begin
1640    printf, lun,idref[j],rafindref[j],decfindref[j],xfindref[j],yfindref[j], magref[j],magerrref[j], skyre
1641    magrun2[j],magerrrun2[j],skyrun2[j],skyerrrun2[j], jmagseltref[j],jerrseltref[j],xstarref[j],ystarref[
1642    cutmax[pp],$
1643    format='(i3,x,d10.6,x,d10.6,x,d10.6,x,d10.6,x,2(f8.3,f8.3,x,f8.3,f8.3),f8.3,f13.3,f8.3,f8.3,x,a20,x,f7
1644 endfor
1645
1646 close, lun
1647 free_lun, lun
1648
1649
1650 zepref    =zpref[0]+25
```

```
zpreferr  =zperrref[0]
zerop2    =zp[0]+25
zpoint2err=zperr[0]


flagrefF=9
nobsrefF=9
framerefF=Framestartref[pp]
NframerefF=Nframeref[pp]
ExpTrefF=ExpTref[pp]
cutrefF=cutref[pp]
cutmaxrefF=cutmaxref[pp]
      x1=raref[pp[0]]
      y1=decref[pp[0]]
      x2=rafindref
      y2=decfindref
      vecdist=sqrt(((x2-x1)*cos(y2*!pi/180.d0))^2+(y2-y1)^2)*3600.0d0
      starpp=where(vecdist eq  min(vecdist) and vecdist lt 1.)
magrefF=magref[starpp]
errrefF=magerrref[starpp]


end





;;;#################
pro  recal, dirin, filer,resid1=resid1,resid2=resid2, cutcal, cutmax, mymagref, myerrref,mymagx,myerrx

print, filer

;resid1or=resid1
;resid2or=resid2

   readcol, filer[0],idref,rafindref,decfindref, xfindref, yfindref, magref,magerrref, skyref,skyerrref,$
   magrun2,magerrrun2,skyrun2,skyerrrun2, jmagseltref,jerrseltref,xstarref, ystarref,namestar,cutmax2,$
   format='(i,d,d,d,d,(f,f,f,f),(f,f,f,f),f,f,d,d,a,f)'

   mystar=where( (xstarref[0]-xfindref)^2+(ystarref[0]-yfindref)^2 eq min((xstarref[0]-xfindref)^2+(ystar
   my2mass=jmagseltref[mystar]

   ;magref=magref+resid1or
   ;magrun2=magrun2+resid2or

;;;@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
;;calibration plot, baset on 2MASS J<15
        !P.CHARSIZE=0.7
        !X.CHARSIZE=0.7
        !Y.CHARSIZE=0.7
        !P.CHARTHICK=3.5
        !X.THICK=4.
        !Y.THICK=4.
        !P.THICK=4.
set_plot, 'ps'
device, filename=filer[0]+'.eps', xs=20.0,  xoff=1.0, yoff=0,/cm,/color

       ;define colors
       yellow = GETCOLOR('yellow', 100)
       red = GETCOLOR('red', 101)
       blue = GETCOLOR('blue', 102)
       green = GETCOLOR('Brown', 103)
       orange = GETCOLOR('Coral', 104)
       darkgreen = GETCOLOR('darkgreen', 105)
       white = GETCOLOR('Snow', 106)
```

```
       gray= GETCOLOR('Slate Gray',107)


;;#######################################################
   la=where(magref lt 99 and magrun2 lt 99 and jmagseltref lt 99)

   ind=where(magref[la] lt cutcal and magref[la] gt cutmax[0])
   res=moment(jmagseltref[la[ind]]-magref[la[ind]])
   ;clip=where(abs(abs(jmagseltref[la[ind]]-magref[la[ind]])-abs(res[0])) lt 2.*sqrt(res[1]) and abs(xsta
   clip=where( ((xstarref[0]-xfindref[la[ind]])^2 + (ystarref[0]-yfindref[la[ind]])^2) gt 15^2 ,nnclip )
   if(filer[0] eq "star72ij283.643208.tab") then   ind=where(magref[la] gt 10 and abs(xstarref[0]^2+ystar

print, nnclip, " nnclip"


if(n_elements(clip) ge 2)  then begin
 res=moment(jmagseltref[la[ind[clip]]]-magref[la[ind[clip]]])
 std=sqrt(res[1])
 stderr=std/sqrt(n_elements([la[ind[clip]]]))
endif
if(n_elements(clip) eq 1)  then begin
 res[0]=jmagseltref[la[ind[clip]]]-magref[la[ind[clip]]]
 std=0.2
 stderr=0.2
endif

resid1=res[0]
;alldet=where(magref gt 0 and magref lt 99)
;magref[alldet]=magref[alldet]+resid1
;magerrref[alldet]=sqrt(magerrref[alldet]^2+stderr^2)
mymagref=magref[mystar]
myerrref=magerrref[mystar]

plot, magref[la], jmagseltref[la]-magref[la], psym=4, xr=[8.5,18],yr=[-0.99,0.99], pos=[0.15,0.43,0.99,0.
ytitle='2MASS J- ANDICAM Mag (run 1)',xstyle=1,ystyle=1

if(clip[0] ne -1) then oplot, magref[la[ind[clip]]], jmagseltref[la[ind[clip]]]-magref[la[ind[clip]]],psy
oplot, [-99,99], [res[0],res[0]]
xyouts,[(!x.crange(1)-!x.crange(0))*0.2+!x.crange(0)],[(!y.crange(1)-!y.crange(0))*0.3+!y.crange(0)], 'ca

oplot, magref[mystar], jmagseltref[mystar]-magref[mystar],psym=4,color=blue, symsize=3
xyouts,[(!x.crange(1)-!x.crange(0))*0.2+!x.crange(0)],[(!y.crange(1)-!y.crange(0))*0.2+!y.crange(0)], 'ta

dd=my2mass-mymagref
xyouts,[(!x.crange(1)-!x.crange(0))*0.2+!x.crange(0)],[(!y.crange(1)-!y.crange(0))*0.1+!y.crange(0)], '2M

;;#######################################################


   la=where(magref lt 99 and magrun2 lt 99 and jmagseltref lt 99)

   ind=where(magref[la] lt cutcal and magref[la] gt cutmax[0])
   res=moment(jmagseltref[la[ind]]-magrun2[la[ind]])
   ;clip=where(abs(abs(jmagseltref[ind]-magrun2[ind])-abs(res[0])) lt 5.0*sqrt(res[1]) and abs(xstarref[0
   ;clip=where( abs(xstarref[0]^2+ystarref[0]^2-xfindref[la[ind]]^2- yfindref[la[ind]]^2) ne min(abs(xsta
   clip=where( ((xstarref[0]-xfindref[la[ind]])^2 + (ystarref[0]-yfindref[la[ind]])^2) gt 15^2 ,nnclip )
   if(filer[0] eq "star72ij283.643208.tab") then   ind=where(magref[la] gt 10 and abs(xstarref[0]^2+ystar

print, nnclip, " nnclip 2"

if(n_elements(clip) ge 2)  then begin
 res=moment(jmagseltref[la[ind[clip]]]-magrun2[la[ind[clip]]])
 std=sqrt(res[1])
 stderr=std/sqrt(n_elements([la[ind[clip]]]))
endif
if(n_elements(clip) eq 1 and clip[0] ne -1)  then begin
```

```
 res[0]=jmagseltref[la[ind[clip]]]-magrun2[la[ind[clip]]]
 std=0.2
 stderr=0.2
endif

resid2=res[0]
;alldet=where(magrun2 gt 0 and magrun2 lt 99)
;magrun2[alldet]=magrun2[alldet]+resid2
;magerrrun2[alldet]=sqrt(magerrrun2[alldet]^2+stderr^2)
mymagx=magrun2[mystar]
myerrx=magerrrun2[mystar]

   plot, magref[la], jmagseltref[la]-magrun2[la], psym=4, xr=[8.5,18],yr=[-0.99,0.99], pos=[0.15,0.71,0.9
   ytitle='2MASS J- ANDICAM Mag (run 2) ',xstyle=1,ystyle=1, title=filer[0]


if(clip[0] ne -1) then oplot, magref[la[ind[clip]]], jmagseltref[la[ind[clip]]]-magrun2[la[ind[clip]]],ps
oplot, [-99,99], [res[0],res[0]]
xyouts,[(!x.crange(1)-!x.crange(0))*0.2+!x.crange(0)],[(!y.crange(1)-!y.crange(0))*0.3+!y.crange(0)], 'ca

oplot, magref[mystar], jmagseltref[mystar]-magrun2[mystar],psym=4,color=blue, symsize=3
xyouts,[(!x.crange(1)-!x.crange(0))*0.2+!x.crange(0)],[(!y.crange(1)-!y.crange(0))*0.2+!y.crange(0)], 'ta
dd=my2mass-mymagx
xyouts,[(!x.crange(1)-!x.crange(0))*0.2+!x.crange(0)],[(!y.crange(1)-!y.crange(0))*0.1+!y.crange(0)], '2M

;;#####################################################

   la=where(magref lt 99 and magrun2 lt 99)
   if(filer[0] eq "star72ij283.643208.tab") then   la=where(magref lt 99 and magrun2 lt 99 and jmagseltre
   plot, magref[la], magref[la]-magrun2[la], psym=4, xr=[7.,18], yr=[-3,3], pos=[0.15,0.15,0.99,0.43],$
   xtitle='ANDICAM Mag(run 1)', ytitle='ANDICAM Mag(run 1)-Mag(run 2)',xstyle=1,ystyle=1,/noerase

   ind=where(magref[la] lt cutcal and magref[la] gt cutmax[0]  and ((xstarref[0]-xfindref[la])^2+(ystarre
   print, ind, "ind"

if(n_elements(ind) ge 2)  then begin
      res=moment(magref[la[ind]]-magrun2[la[ind]])
      clip=where( abs(abs(magref[la[ind]]-magrun2[la[ind]])-abs(res[0])) lt 3.*sqrt(res[1]) )
endif

;star72
ops=where( abs(rafindref-283.643333)*3600.0d0 lt 5 and   abs(decfindref-1.884583) lt 5,ccops)
if ccops gt 0 then begin
   ind=where(magref[la] lt cutcal and magref[la] gt cutmax[0]  and ((xstarref[0]-xfindref[la])^2+(ystarre
   clip=ind
   std=0.2
endif

if(n_elements(clip) ge 2)  then begin
 res=moment(magref[la[ind[clip]]]-magrun2[la[ind[clip]]])
 std=sqrt(res[1])
 stderr=std/sqrt(n_elements([la[ind[clip]]]))
endif
if(n_elements(clip) eq 1 and clip[0] ne -1)  then begin
 res[0]=magref[la[ind[clip]]]-magrun2[la[ind[clip]]]
 std=0.2
 stderr=0.2
endif

print, ind, clip, " ind,clip"
if(clip[0] ne -1 ) then oplot, magref[la[ind[clip]]], magref[la[ind[clip]]]-magrun2[la[ind[clip]]],psym=6
oplot, [-99,99], [res[0],res[0]]
xyouts,[(!x.crange(1)-!x.crange(0))*0.2+!x.crange(0)],[(!y.crange(1)-!y.crange(0))*0.2+!y.crange(0)], 'ca
oplot, magref[mystar], magref[mystar]-magrun2[mystar],psym=4,color=blue, symsize=3
xyouts,[(!x.crange(1)-!x.crange(0))*0.2+!x.crange(0)],[(!y.crange(1)-!y.crange(0))*0.1+!y.crange(0)], 'ta

```

```
;;#########################################################

device,/close
set_plot,'x'

;resid1=resid1or+resid1[0]
;resid2=resid2or+resid2[0]


help, la
help, ind
get_lun, lun
openw, lun, filer[0]+'cal.stars'
for j=0,n_elements(ind)-1 do begin
   printf,lun, rafindref[la[ind[j]]], decfindref[la[ind[j]]], xfindref[la[ind[j]]], yfindref[la[ind[j]]],
   format='(d10.6,x,d10.6,x,f7.2,f7.2,x,f7.3,x,a3)'
endfor
close, lun
free_lun, lun


end



 ;;;#################
pro  recalzero, dirin, filer,resid1=resid1,resid2=resid2, cutcal, cutmax, mymagref, myerrref,mymagx,myerr
print, filer

resid1or=resid1
resid2or=resid2

   readcol, filer[0],idref,rafindref,decfindref, xfindref, yfindref, magref,magerrref, skyref,skyerrref,$
   magrun2,magerrrun2,skyrun2,skyerrrun2, jmagseltref,jerrseltref,xstarref, ystarref,namestar,cutmax2,$
   format='(i,d,d,d,d,(f,f,f,f),(f,f,f,f),f,f,d,d,a,f)'

   mystar=where( (xstarref[0]-xfindref)^2+(ystarref[0]-yfindref)^2 eq min((xstarref[0]-xfindref)^2+(ystar
   my2mass=jmagseltref[mystar]

   mymagref=magref[mystar]+resid1or
   myerrref=magerrref[mystar]
   mymagx=mymagref
   myerrx=myerrref

   la=where(magref lt 99 and magrun2 lt 99 and jmagseltref lt 99)
   ind=where(magref[la] lt cutcal and magref[la] gt cutmax[0] and (xstarref[0]-xfindref[la])^2+(ystarref[

get_lun, lun
openw, lun, filer[0]+'cal.stars'
for j=0,n_elements(ind)-1 do begin
   printf,lun, rafindref[la[ind[j]]], decfindref[la[ind[j]]], xfindref[la[ind[j]]], yfindref[la[ind[j]]],
   format='(d10.6,x,d10.6,x,f7.2,f7.2,x,f7.3,x,a3)'
endfor
close, lun
free_lun, lun

end



;;;-----------------------------------------------TO PERFORM a CROSSCORRELATION
;;;VECTOR NOT POINT
          pro crossid, x1, y1, x2, y2, result, cc

```

```
    ;positional coincidence in IDL
    ;search repetitions of x1 y1 in x2 y2
    result = dblarr(n_elements(x1), 30)

    ;looping trough the stars in the catalog
    test=sqrt(((x2-x1)*cos(y2*!pi/180.d0))^2+(y2-y1)^2)*3600.0d0
    ind=where( test lt 30.d0,cc )
    result=ind


      end


;;;;;;#########################################################
pro   calibsingle,nlog, input, dirref, raxx, decxx, cutcal, getzeropoint, magref, errref, calibrator, myz

;;;;;;#########################################################
readcol, input,   OBJECTref, RAref, DECref, Nframeref, $
ExpTref, DATEref, Framestartref, Framendref, juldayref,secz,cutref,cutmaxref, $
format='(a,d,d,l,f,a,a,a,a,f,f,f)'

factor=cos( DECxx[0]/180.*!pi)
distanza=sqrt(((RAref-raxx[0])*factor[0])^2+(DECref-decxx[0])^2)
pp=where(distanza eq min(distanza) and distanza*60.0d0 lt 2.0,cc)
if(cc gt 1 ) then print, "GOT more than one match, OPS"
if(cc gt 1 ) then stop
print, OBJECTref[pp]

la=pp  ;;;how to select here coordinates
proot=strpos(framestartref[la],'.')
root=strmid(framestartref[la],0,proot+1)
nn=strmid(framestartref[la],proot+1,4)
newind=float(nn+0)
fileref=dirref+root+string(newind[0],f='(f05.0)')+'ave.tab'
print, fileref

readcol,fileref[0], idref, xfindref,yfindref,magrefx,magerrrefx, skyref,skyerrref, $
nframeref, xstarref,ystarref,starnameref, $
format='(i,f,f,f,f,f,f,i,f,f,a)'
distanza=sqrt((xfindref-xstarref[0])^2+(yfindref-ystarref[0])^2)
selstar=where(distanza eq min(distanza) and distanza lt 10)
print, distanza, "dist", selstar

if(nlog eq 1) then  masterx="master_sort1.tab"
if(nlog eq 2) then  masterx="master_sort2.tab"
if(nlog eq 3) then  masterx="master_sort3.tab"

;make master table
readcol, masterx,  OBJECTref, RAref, DECref, flag1, frame1, Nframeref, ExpTref,$
cutref,cutmaxref,magref, errref, $
flag2, frame2, file2,  Nframe2, ExpT2, zeropointref2, zeropoint2, mag2, err2,$
flag3, frame3, file3,  Nframe3, ExpT3, zeropointref3, zeropoint3, mag3, err3,$
format='(a,d,d,i,a,i,f,f,f,f,f,(i,a,a,i,f,f,f,f,f),(i,a,a,i,f,f,f,f,f))'

;;flag1 flag(0,*)
;;flag2 flag(1,*)
;;frameq=frame(nlog-1,*)

if(nlog eq 1) then  frameq=frame1
if(nlog eq 2) then  frameq=frame2
if(nlog eq 3) then  frameq=frame3
frameq=strmid(frameq,0,11)
if(nlog eq 1) then  fileq=file2
if(nlog eq 2) then  fileq=file2
if(nlog eq 3) then  fileq=file3
if(nlog eq 1) then  zeroq=zeropointref2
```

```
if(nlog eq 2) then  zeroq=zeropoint2
if(nlog eq 3) then  zeroq=zeropoint3

pos=strpos(fileref[0], 'binir')
root=strmid(fileref[0],pos,11)
print, starnameref[selstar], " mystar"
print, magrefx[selstar], " mag instrumenta mystar"

if(calibrator[0] ne 'none') then begin
  la=where(root eq frameq and zeroq lt 0 and fileq ne 'none' and objectref eq calibrator[0],nncount)
  getzeropoint=mean(zeroq[la])
endif
if(calibrator[0] eq 'none' and myzero lt 0.0) then getzeropoint=myzero[0]
  print, getzeropoint, "selected zeropoint"
  magref=magrefx[selstar]+getzeropoint
  errref=magerrrefx[selstar]
  print, magref, "  stellar magnitude calibrated with the selected zeropint or calibrator"

end
```