
Subject: Re: adjusting brightness of an image
Posted by [David Foster](#) on Tue, 20 Oct 1998 07:00:00 GMT
[View Forum Message](#) <> [Reply to Message](#)

Anil Kochhar wrote:

>
> Hi,
> I am writing a program which scales images(i.e. images of galaxies) ,
> adjusts their brightness and apparent distances, and then adds each image
> to a final image, which in the end just comprises all of the images which
> were added to it.
>
> Many of the images on the final image appear very faint, and I was told
> that
> one can use the tvscl command to adjust brightness by using the following
> syntax:
>
> tvscl<Number1>Number2
>

Anil -

You might find the following routine ADJUST_INTENSITY.PRO useful.
It allows you to adjust the intensity of one image relative to
another, and displays histograms of both frequency distributions.
Following the routine is a .doc documentation file. Hope this helps.

Dave

```
===== ADJUST_INTENSITY.PRO =====  
; ADJUST_INTENSITY.PRO 3-11-98 DSFoster  
;  
; Function to allow user to interactively adjust the "brightness"  
; of an image in a given window, using a slider, and return the  
; correction factor chosen by the user.  
;  
; Updates:  
;  
; 4-27-95 DSF Use two sliders instead of one to adjust the plot maxima  
; for the histogram distributions. Could not use one maximum to  
visualize  
; both simultaneously.  
; 8-28-95 DSF Maintain original image (unadjusted) from window when  
this  
; routine is first called, and make all adjustments to this image.  
Fix  
; bug that screwed up background in image.  
; 11-01-95 DSF Remove RETAIN=2 from main widget_draw calls to try to fix
```

```
; problem with image not being updated. Notify user of invalid
return
; from TVRD() .
; 11-30-95 DSF Add argument checking.
; 5-14-97 DSF Update for IDL 5.0.
; 3-11-98 DSF Add optional fifth argument which is the original
; (uncorrected) image displayed in the window to be adjusted.
; This avoids losing information in pixels whose values get
; truncated when the correction factor has already been applied.
```

```
;-----
; procedure ADJUST_INTENSITY_UPDATE_IMAGE
;
; Procedure to apply correction factor(s) to update the image.
;
; Do NOT use TVRD to read image in case it's scrollable;
; there's a bug in the TVRD() function! (Read into temporary pixmap,
then
; use TVRD.)
;
; THIS MUST BE FIRST ROUTINE: COMMON BLOCK DEFINED!
;-----
```

```
PRO adjust_intensity_update_image, image_arg, GET_IMAGE=get_image, $
    NO_DISPLAY=no_display
```

```
common adjust_intensity_common, d, hist1, hist2, window_image
```

```
old_window = !d.window
```

```
if (keyword_set(GET_IMAGE)) then begin
    wset, d.pixmap
    DEVICE, COPY=[0,0, d.window_size(0), d.window_size(1), 0,0,
d.adjust_window]
    image = tvrd()
    info = size(image)
    if (total(info(1:2) ne d.window_size(0:1)) gt 0) then begin
        image_arg = -1          ; Image not correct dimensions
    endif else begin
        image_arg = image
    endelse
endif else begin
    image = window_image
endelse
```

```
if (not keyword_set(NO_DISPLAY)) then begin
    r = where(window_image ge byte(d.minval) and window_image le
```

```

byte(d.maxval))
  if (r(0) ne -1) then image(r) = $
    byte( (d.minval > (window_image(r) + d.correction)) < d.maxval)
  wset, d.adjust_window
  tv, image ; Redisplay
image
endif
if (old_window ne -1) then wset, old_window

return
END

;-----
; procedure ADJUST_INTENSITY_PLOT
;
; Procedure to plot histogram distributions in draw windows.
;-----

PRO adjust_intensity_plot, PLOT_FIRST=plot_first,
PLOT_SECOND=plot_second

common adjust_intensity_common

old_window = !d.window

if (keyword_set(PLOT_FIRST)) then begin
  wset, d.draw_win(0)
  plot, hist1, xrange=[0,d.xplot_max], yrange=[0,d.yplot_max(0)], $
    color=d.plot_color
endif

if (keyword_set(PLOT_SECOND)) then begin
  hist = shift(hist2, d.correction)
  n = n_elements(hist)
  if (d.correction lt 0) then begin ; Remove wrap-around
    hist((0 > (n+d.correction-1)) < (n-1) : n-1) = 0
  endif else if (d.correction gt 0) then begin
    hist(0 : 0 < ((d.correction-1) < (n-1))) = 0
  endif
  wset, d.draw_win(1)
  plot, hist, xrange=[0,d.xplot_max], yrange=[0,d.yplot_max(1)], $
    color=d.plot_color
endif

if (old_window ne -1) then wset, old_window
return
END

```

```

;-----
; procedure ADJUST_INTENSITY_EVENT
;
; Event handler.
;-----

```

PRO adjust_intensity_event, event

common adjust_intensity_common

```

widget_control, event.id, get_uvalue=user_value
if (n_elements(user_value) eq 0) then user_value = ""
type = strmid(tag_names(event, /structure), 7, 1000)

```

```

case (type) of
  "BUTTON": begin
    case (user_value) of
      "DONE": begin
        wdelete, d.pixmap
        WIDGET_CONTROL, event.top, /DESTROY
      end
      "HELP": begin
        WIDGET_CONTROL, event.top, /HOURGLASS
        LHELP, 'adjust_intensity.doc', group=event.top
      end
      "CANCEL": begin
        if (d.orig_correction ne -1) then begin
          d.correction = d.orig_correction
        endif else begin
          d.correction = 0
        endelse
        if (d.correction ne d.last_correction) then begin
          WIDGET_CONTROL, event.top, /HOURGLASS
          ADJUST_INTENSITY_UPDATE_IMAGE
        endif
        d.correction = -5000 ; Return error

```

code

```

    wdelete, d.pixmap
    WIDGET_CONTROL, event.top, /DESTROY
  end
else:
endcase
end
"SLIDER": begin
  case (event.id) of
    d.brightness_slider: begin
      case (event.drag) of
        0: begin ; Update image (slider released)

```

```

        WIDGET_CONTROL, event.top, /HOURGLASS
        ADJUST_INTENSITY_UPDATE_IMAGE
        d.last_correction = d.correction ; Record last
factor
        end
    1: begin ; Update plot (slider dragged)
        d.correction = event.value
        ADJUST_INTENSITY_PLOT, /plot_second
    end
endcase
end
d.xplot_slider: begin
    d.xplot_max = event.value
    ADJUST_INTENSITY_PLOT, /plot_first, /plot_second
end
d.yplot_slider(0): begin
    d.yplot_max(0) = event.value
    ADJUST_INTENSITY_PLOT, /plot_first
end
d.yplot_slider(1): begin
    d.yplot_max(1) = event.value
    ADJUST_INTENSITY_PLOT, /plot_second
end
endcase
end
else:
endcase

return
END

;-----
; function ADJUST_INTENSITY
;
; Main function.
;-----
FUNCTION adjust_intensity, image, xdim, ydim, adj_window, adj_image, $
    TITLE=title, PARENT=parent, MINIMUM=minimum, MAXIMUM=maximum, $
    INIT=init, MAX_SUBTRACT=max_subtract, MAX_ADD=max_add,
MODAL=modal, $
    POSITION=position, COLOR=color, IGNORE=ignore

common adjust_intensity_common

d = { base: 0L, $ ; Main widget base
    draw: LONARR(2), $ ; Draw widgets for histogram plots
    brightness_slider: 0L, $ ; Slider to adjust correction factor

```

```

yplot_slider: LONARR(2), $ ; Sliders for y-range of plots
xplot_slider: 0L, $ ; Slider for x-range of plots
draw_win: LONARR(2), $ ; Window IDs for plot draws
adjust_window: -1L, $ ; Window with image to adjust
pixmap: -1L, $ ; Pixmap for use with TVRD()
window_size: INTARR(2), $ ; Size of window
xplot_max: 0L, $ ; X-range for plots
yplot_max: LONARR(2), $ ; Y-range for plots (one each image)
plot_color: !d.table_size, $; Color to use for plots
minval: 0, $ ; Minimum value to change in image
maxval: 0, $ ; Maximum value to change in image
orig_correction: -1, $ ; Original correction passed with
INIT kw
    last_correction: 0, $ ; Last correction made to image
    correction: 0 } ; Correction added to image

; Check arguments

info = size(image)
if (info(0) ne 2) then $
    message, 'Argument one must be two-dimensional array'

device, window_state=open_windows
if (open_windows(adj_window) ne 1) then $
    message, 'Argument four not a valid window ID'

if (xdim le 0 or ydim le 0) then $
    message, 'Arguments two and three must be positive and nonzero'

; Initialize variables

d.adjust_window = adj_window
d.window_size = [xdim, ydim]
d.xplot_max = max(image) + 50

if (keyword_set(COLOR)) then d.plot_color = color
if (keyword_set(INIT)) then begin
    d.correction = init
    d.last_correction = init
    d.orig_correction = init
endif
if (not keyword_set(TITLE)) then title = 'Adjust Image Intensity'
if (not keyword_set(MAX_SUBTRACT)) then max_subtract = -100
if (not keyword_set(MAX_ADD)) then max_add = 100
if (not keyword_set(MINIMUM)) then minimum = 0
if (not keyword_set(MAXIMUM)) then maximum = !D.TABLE_SIZE - 1
d.minval = minimum
d.maxval = maximum

```

```

if (keyword_set(POSITION)) then begin
    if (n_elements(position) ne 2) then $
        message, 'Keyword POSITION must be 2-element array'
    rel_pos = [position(0), position(1)]
endif else begin
    rel_pos = [.333, .250]
endelse

if (keyword_set(PARENT)) then begin                ; Center of
program
    valid_parent = WIDGET_INFO(parent, /valid_id)
    if (valid_parent ne 1) then begin
        message, 'Keyword PARENT not a valid widget ID', /continue
        goto, USE_SCREEN_SIZE
    endif
    WIDGET_CONTROL,parent,TLB_GET_OFFSET=p_offset
    WIDGET_CONTROL,parent,TLB_GET_SIZE=p_size
    xpos = p_offset(0) + p_size(0) * rel_pos(0)
    ypos = p_offset(1) + p_size(1) * rel_pos(1)
endif else begin                ; Center of
screen
    USE_SCREEN_SIZE:
    valid_parent = 0
    DEVICE, get_screen_size=s_size
    xpos = s_size(0) * rel_pos(0)
    ypos = s_size(1) * rel_pos(1)
endelse

if ( valid_parent eq 1 and keyword_set(MODAL) ) then begin
    d.base = WIDGET_BASE(/column,xpad=10,ypad=10,title=title,
xoffset=xpos, $
        yoffset=ypos, group_leader=parent, /modal)
endif else begin
    d.base = WIDGET_BASE(/column,xpad=10,ypad=10,title=title,
xoffset=xpos, $
        yoffset=ypos)
endelse

junk = WIDGET_BASE(d.base, /frame, /column)
junk2 = WIDGET_BASE(junk, /row)
d.yplot_slider(0) = WIDGET_SLIDER(junk2, minimum=0, maximum=2000, $
    value=d.yplot_max(0), /vertical, /suppress_value, /drag)
d.draw(0) = WIDGET_DRAW(junk2, xsize=260, ysize=120)

junk2 = WIDGET_BASE(junk, /row)
d.yplot_slider(1) = WIDGET_SLIDER(junk2, minimum=0, maximum=2000, $
    value=d.yplot_max(1), /vertical, /suppress_value, /drag)
d.draw(1) = WIDGET_DRAW(junk2, xsize=260, ysize=120)

```

```
d.xplot_slider = WIDGET_SLIDER(junk, minimum=1, maximum=2000, $  
    value=d.xplot_max, /drag)
```

```
junk = WIDGET_BASE(d.base, /frame, /column)  
junk2 = WIDGET_LABEL(junk, value='Signal Intensity Adjustment')  
d.brightness_slider = WIDGET_SLIDER(junk, xsize=280,  
    minimum=max_subtract, $  
    maximum=max_add, value=d.correction, /drag)
```

```
junk = WIDGET_BASE(d.base)  
junk2 = WIDGET_BASE(d.base, /row, /frame)  
done_button = WIDGET_BUTTON(junk2, value='Done', uvalue='DONE')  
junk3 = WIDGET_BUTTON(junk2, value='Cancel', uvalue='CANCEL')  
junk3 = WIDGET_BUTTON(junk2, value='Help', uvalue='HELP')
```

```
WIDGET_CONTROL, d.base, /REALIZE  
WIDGET_CONTROL, d.draw(0), get_value=temp_window  
d.draw_win(0) = temp_window  
WIDGET_CONTROL, d.draw(1), get_value=temp_window  
d.draw_win(1) = temp_window
```

```
; Create pixmap window to use with TVRD()
```

```
old_window = !d.window  
window, xsize=d.window_size(0), ysize=d.window_size(1), /free, /pixmap  
d.pixmap = !d.window  
if (old_window ne -1) then wset, old_window
```

```
; Compute histogram of images and plot histograms
```

```
; Image passed as argument
```

```
hist1 = histogram(image)  
hist1(0 : 0 > (d.minval-1)) = 0  
n1 = n_elements(hist1)  
hist1((d.maxval+1) < (n1-1) : n1-1) = 0  
zoom = long(long(xdim) * ydim) / n_elements(image) ; Account for  
image/window  
if (zoom ne 1) then hist1 = hist1 * zoom ; size mismatch
```

```
; Read image from window. Save copy in common-block variable.  
; Alternatively, if the original was passed as 5th argument then  
; use this to avoid losing information when pixel values get  
; "truncated" when overflow occurs when applying the correction  
; factor.
```

```

if (n_params() ge 5) then begin
  info = size(adj_image)
  if (info(0) ne 2) then begin
    message, 'Argument 5 must be 2D array'
  endif else begin
    adj_image_supplied = 1
    if (info(1) ne xdim or info(2) ne ydim) then begin
      image2 = congrid(adj_image, xdim, ydim) ; New dimensions
    endif else begin
      image2 = adj_image
    endelse
    window_image = image2 ; Save original image
    ind = where(image2 eq 0)
    image2 = byte( (d.minval > (temporary(image2) + d.correction)) <
$
      d.maxval )
    if (ind(0) ne -1) then $
      image2(ind) = 0B ; Restore "background"
    endelse
  endif else begin
    adj_image_supplied = 0
    ADJUST_INTENSITY_UPDATE_IMAGE, image2, /get_image, /no_display
    if (image2(0) eq -1) then begin
      message, 'Invalid return from TVRD() function', /continue
      print, ' (notify programmer)'
      message, 'Error reading from IDL window.'
    endif
    window_image = image2
  endelse

; If initial correction factor specified and original image was NOT
; supplied then reverse the correction to get the original image.
; Note that if the original image is not used then this can result
; in loss of information for pixels that were truncated by application
; of correction factor.

if (adj_image_supplied eq 0 and keyword_set(INIT)) then begin
  r = where(image2 ge byte(d.minval) and image2 le byte(d.maxval))
  if (r(0) ne -1) then $
    window_image(r) = byte( (d.minval > (image2(r) - init)) <
d.maxval )
endif

hist2 = histogram(image2)
n2 = n_elements(hist2)
hist2(0 : 0 > (d.minval-1)) = 0
hist2((d.maxval+1) < (n2-1) : (d.maxval+1) > (n2-1)) = 0
hist2 = shift(hist2, -d.correction) ; Need uncorrected

```

```

distribution
hist2(0 : 0 > (d.minval-1)) = 0
hist2((d.maxval+1) < (n2-1) : (d.maxval+1) > (n2-1)) = 0

d.yplot_max(0) = max(hist1) < 32766
d.yplot_max(1) = max(hist2) < 32766
if (d.yplot_max(0) le 0 or d.yplot_max(1) le 0) then begin
    message, 'Images must contain only positive values', /continue
    widget_control, d.base, /destroy
    return, -5000
endif

for i = 0, 1 do begin
    widget_control, d.yplot_slider(i), set_slider_max=d.yplot_max(i)*2
    widget_control, d.yplot_slider(i), set_value=d.yplot_max(i)
    widget_control, done_button, /input_focus
endfor

ADJUST_INTENSITY_PLOT, /plot_first, /plot_second

XMANAGER, "adjust_intensity", d.base, /no_block

return, d.correction      ; Return adjustment made to image
END

```

===== ADJUST_INTENSITY.DOC =====

ADJUST_INTENSITY

This routine allows you to interactively adjust the signal intensities of one image so that it can be viewed simultaneously with another image, using the same grayscale settings.

Appearing in two windows are plots of the histograms for each image. These make it easy to adjust the intensities of one image so that they are comparable with the other. To do so, drag the slider under "Signal Intensity Adjustment" to see how the distribution of the second image is changed by your adjustments. When you release the slider the results will be displayed in the window containing the second image.

The adjustment you are making is simply to add a constant value (positive or negative) to signal values in the image.

Note that you can change the X and Y range for the plots using the horizontal and vertical sliders next to the plot windows, to ensure that the entire distributions may be viewed. There are

two sliders for the Y ranges, so they may be adjusted independently.

Select "Done" when you are satisfied with a particular adjustment, or "Cancel" if you wish to abort the adjustment. You can select "Help" for help information (this file).

The remaining information in this file concerns how to call this routine in a program.

Notes

The situation in which ADJUST_INTENSITY will be called will be the following: there are two windows in which two images are displayed, and the user needs to be able to adjust the signal intensities of the second image to make them more like the first image.

You must supply as arguments: (1) the image displayed in the first window; (2) and (3) the X- and Y-dimensions of the second window; and (4) the window ID for the second window.

It is important that the contents of the second window are not changed by any application other than ADJUST_INTENSITY until this widget returns. See the keyword MODAL below.

Calling Sequence

```
Rtnval = ADJUST_INTENSITY(Image, Xdim, Ydim, Window [, Adj_Image])
```

Arguments

Adj_Image

This optional argument contains the original image used to display the image in the window which is to be adjusted. Even if the intensity of the image in this window has been adjusted previously, Adj_Image will be uncorrected.

Passing this argument avoids the problem with losing pixel information when values are truncated from previous corrections. This routine can always use this original image and correct *that*.

Note that the image passed must be scaled to fit within a byte image. You can use BYTE_SCALE() for this purpose. Also note that the background must be preserved after the scaling.

Image

The displayed image whose signal values you will be attempting to approximate by adjusting the OTHER image. This must be the actual image displayed in the window (not the image used in a call to TVSCL, which will display the scaled image).

It does not matter whether the image has been zoomed up before display or not; the routine will account for this. You can always pass the original image if you like.

Xdim, Ydim

The X and Y dimensions of the window (argument Window) which contains the image whose signal intensities you will be adjusting.

These are NOT the dimensions of the image represented by the argument Image.

Window

The window identifier of the window containing the image whose signal intensities you will be adjusting.

Keywords

COLOR

The color used to plot the histograms. If not supplied defaults to the value of !D.TABLE_SIZE.

INIT

Set this to an initial value for the correction factor to be added to the image. If not supplied then defaults to zero (no correction). This is useful in situations where this routine may have been called previously, and the correction factor is being used by the calling module.

MAX_ADD

Set this to the maximum positive adjustment allowed for the image. This will be the most you can add to the signal intensities.

MAX_SUBTRACT

Set this to the minimum positive adjustment allowed for the image.

This will be the most you can subtract from the signal intensities.

MAXIMUM

The maximum pixel value to consider. Only signal values less than or equal to this will be plotted in the histogram, and only these

pixels

will be adjusted in the second window when an adjustment is applied.

This keyword is useful when the calling module has reserved colors.

Defaults to !D.TABLE_SIZE.

MINIMUM

The minimum pixel value to consider. Only signal values greater than

or

equal to this will be plotted in the histogram, and only these

pixels

will be adjusted in the second window when an adjustment is applied.

This keyword is useful when the calling module has reserved colors.

Defaults to 0.

MODAL

Set this to make this a modal widget; all other widgets will be insensitive until this widget returns. It is especially important to use this when the second window can be altered by operations in the calling program (the contents must not be changed by another application!).

PARENT

Set this to the top-level-base of the calling program to have the widget positioned relative to the program itself. Otherwise the position is relative to the screen.

POSITION

This keyword lets you adjust where the widget is located relative to either the calling program or the screen (see PARENT keyword). Set this to a FLTARR(2) containing the proportional X- and Y-positions of the widget. Example: POSITION=[0.5,0.5] will put the upper-left corner of the widget at the center of either the calling program or the screen.

TITLE

Use this to set the title of the widget. Defaults to "Adjust Image Intensity".

Outputs

Selecting the "Done" button causes the current correction factor to be returned. If "Cancel" is selected then -5000 is returned.

The image displayed in the second window identified by the argument Window is changed (brightened or darkened) whenever the user releases the slider labelled "Signal Intensity Adjustment" after a drag operation. If "Cancel" is selected then the original image is displayed before returning.

Example

The following is the call made to ADJUST_INTENSITY within the MrRegion program:

```
; Pass original (uncorrected) image from window to adjust

adj_image = byte_scale( images(*,*,eq_image), $
    bottom=col.min_graylevel, $
    max=d.image_max(eq_image) )
adj_image = adj_image * (byte_images(*,*,0) gt 0B) ; Background!

temp = BYTE_SCALE(images(*,*,0), bottom=d.min_graylevel, $
    max=d.image_max(0))

; Supply corrected image if it has itself already been adjusted

if (d.image_correction(1-eq_image) ne 0) then begin
    temp = (col.min_graylevel > (temporary(temp) + $
        d.image_correction(1-eq_image))) < (!D.TABLE_SIZE-1)
endif
temp = temp * (byte_images(*,*,0) gt 0B) ; Background to zero

ret = adjust_intensity(temp, 512,512, adj_window, adj_image, $
    minimum=d.min_graylevel, init=d.image_correction, $
    parent=event.top, max_subtract=-100, max_add=100, $
    title='MrRegion : Adjust Image Intensity', /modal, $
    position=[0.059, 0.20])
if (ret ne -5000) then d.image_correction = ret
```

Note that the image passed as argument is scaled first (since this is what is displayed in the window). Also, the background value of zero has to be preserved, since scaling screws this up. Both of these statements must also be true if the optional 5th argument

Adj_Image is passed.

Colors for pixel values below D.MIN_GRAYLEVEL are preserved. Notice that the correction factor used in MrRegion is updated only if the user did NOT press "Cancel" to quit.

=====

Dave

--

~~~~~  
David S. Foster      Univ. of California, San Diego  
Programmer/Analyst   Brain Image Analysis Laboratory  
foster@bial1.ucsd.edu   Department of Psychiatry  
(619) 622-5892      8950 Via La Jolla Drive, Suite 2240  
                         La Jolla, CA 92037  
~~~~~
