Once again, to execute this file, I just start up the **skyview** program, and type:

```
> ta lmc.tak
```

### 2.4.7 Coordinate Pick

To get values and coordinate readouts of pixels in an image displayed on the screen, use the **pick** command. Two alternatives exist for use of the **pick** command: Interactive and command line input. Both alternatives place a green[8] crosshair on the pixel and report the pixel value and coordinate location. Another command you might want to use is the **mark**[9] command. This function simply puts up a marker (of a given symbol shape, size and color) on a known position. The **mark** command identifies locations as closely as possible, while **pick** identifies the nearest pixel location (and coerces the crosshair to the center of the pixel). **mark** doesn't report data values, but is especially useful when dealing with catalogs or table input.

**Interactively picking coordinates.** Put the cursor into an image window, and click the middle button on the mouse.

**or**

Type the **pick** command:

```
> pi
```

You now have interactive control of the cursor. The buttons on your mouse are now redefined for use with the **pick** command:

| command: | left button: | center button: | right button: |
|---|---|---|---|
| pi | pick | end | — |

Move the cursor around by simply moving the mouse. When you are at the position you desire information about, click the left button and you will get the flux and location of that pixel. To exit the **pick** command, click the middle button.

[8]See **Section 2.4.3** for instructions on how to **set** a different default color for **skyview** commands that overlay graphics on an image.

[9]See **Section 2.5.2** for details on the use of the **mark** command.

**Coordinate Pick using Command Line Input.** Use the **pick** command to find out where a specific location is on the image.

```
> pi 5h32m56.8s 9d40m
```

This puts a cursor on the given coordinate, and reports the flux. *Don't like Equatorial coordinates?* The cs command will change to the coordinate system of your choice[10]:

```
> cs gal
```

changes to Galactic Coordinates. The **pick** command will also average over $n$ x $n$ pixels if you specify an integer as the first (or only) parameter:

```
> pi 5
```

will give interactive control of the cursor with the mouse, and report the average and maximum values in a 5x5 box around each position you pick.

```
> pi 5 5h32m56.8s 9d40m
```

will report the average and maximum values of the 25 pixels centered on the position given.

### 2.4.8 Slice

To get a look at the profile of the data values along a line (cut a slice) through an image on the screen, you have two options: Interactive and command line input. Both options give the same 3 output products:

- A profile of the data along the cut is plotted on the image display[11].
- The minimum, maximum and standard deviation values on the cut are reported in the text window running **skyview**.
- A file named **slice** which contains the flux values and coordinates for each point on the cut is created in the current directory.

[10]See **Section 2.4.1** for more information on the use of the **cs** command

[11]By default, the cut is displayed in green, and the profile is red. See **Section 2.4.3** for instructions on how to **set** a different default overlay color

**Interactive Slicing.** Put the cursor into the image you want to slice, and use the **slice** command to interactively manipulate the length and direction of the slice.

```
> sl
```

The buttons on your mouse are now redefined for use with the **slice** command:

| command: | left button: | center button: | right button: |
|---|---|---|---|
| sl | center | end | end points |

Simply moving the mouse around causes the line to move without changing it's angle or length. Pressing the right button freezes the center point of the line and allows you to change the length and angle by moving the mouse. You can change the position, angle, and length of the line as much as you want by iterating between the left and right mouse buttons. Once you are satisfied with the position and angle of the cut, click the middle button.

**Command Line Input of the Slice Command.** Use the **slice** command to plot the slice between two known positions.

```
> sl 2h54m41s 61d23m54s 2h36m38.9s 59d31m32s
```

*Don't want to write over the* **slice** *file each time?* Just give a new file name.

```
> sl slicefile.2
```

**or**

```
> sl slicefile.2 2h54m 61d23m54s 2h36m 59d31m
```

### 2.4.9 DefineArea/Examine

To calculate the flux over an area on a displayed image, you can use the **define_area** and **examine** commands. The **background** command is used if you wish to subtract the local background in the calculation. The three steps are:

Define the background to be subtracted
Define the area
Examine the area

**Define the Background to be Subtracted.** There are 3 ways to select the background level to be subtracted from each pixel when examining areas:

Interactively picking points
Using the value of a given point.
Declaring a particular value to be used.

**Interactive Selection.** The interactive option of the **background** command works just like the **pick** command.

```
> ba
```

This will give you interactive control of the cursor. You now pick the points you want the program to average as the background flux.

| command: | left button: | center button: | right button: |
|---|---|---|---|
| ba | pick | end | — |

Move the cursor around by simply moving the mouse. When you are at a position you wish to use as a background point, click the left button and you will get the flux and location of that pixel. Once you have selected the pixels you want, click the middle button and the average will be computed and used as the background.

**Setting the Flux Found at a Particular Coordinate.** Use the flux of a specific location as the background level.

```
> ba 5h32m56.8s 9d40m
```

This puts a cursor on the given coordinate, reports the flux and uses it as the background. The **ba** command will also average over $n$ x $n$ pixels by giving the integer to use as the first (or only) parameter in either of the two previous situations:

```
> ba 5
```

will give interactive control of the cursor with the mouse, and report the average and maximum values in a **5x5** box around each position you pick. Once you click the middle button to end the ba function, all the averages get averaged and used as the background level.

```
> ba 5 5h32m56.8s 9d40m
```

will report the average and maximum values of the 25 pixels around the position given and use that average as the background level.

**Setting the Background Level to a Previously Determined Level.**

```
> ba = 4.32561e6
```

**or**

```
> ba = 0
```

The background level is 'sticky', so the same value will be used on **all** images every time you examine an area, until you change it.

**Defining the Area.** First, a brief look at the options for the **define area** command:

- The shapes you can define are circles, boxes, ellipses and polygons.
- You can create the shapes with the mouse interactively, with a list of known coordinates or a combination of the interactive and list modes.

**Interactively Defining Areas.** To invoke the interactive mode:

```
> da
```

Enables the mouse so that you may interactively outline the area in the default shape of a circle. This achieves the same outcome as

```
> da ci
```

To define a rectangular-shaped area

```
> da bo
```

To define a polygon

```
> da po
```

In each of the above cases, the buttons on your mouse are now redefined:

| command: da bo | left button: center | center button: end | right button: size |
|---|---|---|---|
| da ci | center | end | size |
| da po | corner | end | |
| da el | center | end | size/angle |

When defining a rectangle with the **bo** option, clicking on the right mouse button allows further mouse movement to change the size of the box. Clicking on the left mouse button holds the size of the box and allows further mouse movement to move the box around on the image.

To define a circular area, use the buttons in the same way.

To define an ellipse, clicking the right button allows further mouse movement to change the size of the ellipse. Clicking the right button again (i.e., two clicks of the right mouse button in a row) will freeze the size, and allow you to change the angle of the ellipse as you move the mouse. Clicking the left mouse button holds both the shape and the angle of the ellipse, and moves the ellipse around on the image as the mouse roams.

With any of these three shapes, you can switch back and forth between the right and left button functions as often as you like, until you have the area defined as you like it. Once you're happy with it, click the middle mouse button or press the 'Return' key on the keyboard to end.

To define a polygon, each click of the left mouse button places a vertex of the polygon at the current cursor location. When you have defined the last vertex, and wish to close the polygon, either click the middle mouse button or press the 'Return' key on the keyboard.

Once you have defined an area, you will get the same shape for all subsequent areas you wish to define, if you don't use any parameters with the **da** command. You can define multiple areas by numbering them.

```
> da 1 bo
```

will cause the box you are about to define to be named **Area #1**. **skyview** will keep track of up to 50 areas.

**Command Line Input for Defining Areas** If you want to define a circular or elliptical area, and you know the exact coordinates and size, you can enter the command explicitly.

```
> da ci 5 5h49m8s 9d48m
```

would define a circular area with a radius of 5 arcminutes centered at 5h49m8s 9d48m (When using IRAF or image coordinates, the radius specified is in number of pixels, not arcminutes).

To define an ellipse in this manner, there are 5 parameters: The $x$ and $y$ coordinates of the center of the ellipse; the width (delta x, or horizontal) and the height (delta y, or vertical) for the size of the ellipse; and the rotation angle of the ellipse. For example:

```
> da el 0h40m05.5s 41d01m32s 30 95 75
```

would create an ellipse centered at 0h40m05.5s 41d01m32s, that is 30 arcminutes wide, 95 arcminutes high, and rotated 75 degrees counter-clockwise (When using IRAF or image coordinates, the width and height specified are given in number of pixels, not arcminutes).

**Note**: If you're used to thinking of ellipses in terms of minor and major axes and rotation angle of major axis, then enter their values in **that** order. The rotation angle is measured from the vertical on the image. If 'up' is north, and 'left' is east, the rotation angle corresponds to astronomical position angle.

**The Combo.** To invoke the combination mode:

```
> da ci 5
```

Creates a circle with a **radius** of 5 arcminutes and enables the mouse to move the circle to any position on the screen. Note: The **radius** for circles defined in images with non-square pixels is accurate in the horizontal direction (The radius is measured along the horizontal axis).

**Defining Areas Using a List of Coordinates.** To invoke the list mode:

```
> da li
```

The list option of **da** enables you to create any of the three shapes by manually entering the proper coordinate and size input (set the coordinate system using the **cs** command). For example, if you want to define a circle 10 arcminutes in diameter around your favorite object:

```
> da li
```

The list option then prompts you for the proper information with a special prompt:

```
define_area>
```

The first prompt needs to know the shape, and if it's a circle, what the radius (in arcminutes) of the circle shall be.

```
define_area> ci 5
```

means the shape you want is a circle and it should have a radius of 5 arcminutes. Next you give the center position of the circle.

```
define_area> 5h49m8s 9d48m
```

To define a polygon using some boundaries that you know of:

```
> da li
define_area> po 8
```

means the shape you want is a polygon and it will have at the most, 8 vertices. You would then be prompted 8 times. Each prompt is for a vertex.

```
define_area> 5h33m4.9s -4d24m
define_area> 5h34m -4d46m
define_area> 5h34m30s -5d
define_area> 5h35m -5d30m
define_area> 5h30m30s -5d5m
define_area> 5h30m -4d30m
define_area> 5h30m -4d13m30s
define_area> 5h32m20s -4d16m
```

Once you have put in the last point, data input is automatically terminated, and the shape is closed. **If you don't know how many points will be made**, put in some high number and when you get as many drawn as needed, type **e** (for end) <CR> and input will be ended. Note: The **radius** for circles defined in images with non-square pixels is accurate in the horizontal direction (The radius is measured along the horizontal axis).

### Examining the area.

```
> ex
```

examines the last area defined. Reports the minimum, maximum, mean, rms, flux integral, steradian area, number of pixels, centroid and flux weighted centroid of the area.

```
> ex 1
```

examines the area you defined as **Area #1**, in the **define area** example above. *Want to examine the same area in another image?* Just paint the other image and examine!

```
> pa band1;da ci;ex
> pa band2;ex
```

Or, if the image **band2** were already painted, you could use the **current image** (**ci**) command to change to that image, and then examine:

```
> pa band2
> fr adv
> pa band1;da ci;ex
> ci band2;ex
```

*Want to re-examine an area previously defined?* The **change area** command lets you call back areas you've already defined, providing you numbered them when you defined them.

```
> ca 10;ex
```

**or**

```
> ex 10
```

will re-draw the graphics for previously defined **Area #10** on the image and examine the area again. This can be done on a different image than the first time, as long as the area is covered in the new image. Note that the defined areas are remembered using **image** coordinates (line and sample); so, for the **examine_area** to be meaningful on a second image, the two images must have the same projection parameters.

#### 2.4.10 Skybox

The **skybox** command can be used to:

- measure sky value
- measure local frame flatness
- measure the pixel to pixel RMS
- measure the 'true' sky error (sigma of the mean)

The skybox command uses interactive selection (via the mouse) of square areas on an image to gather sky statistics. The syntax is

```
> sb n cliplevel
```

The 'n' parameter is used to specify the size of the area. The 'cliplevel' parameter is used to specify what dispersion of the pixel values to use in the computations. Each time the left button is clicked, an $n \times n$ pixel box is drawn. The following information is reported on all the pixels within in the box whose value falls within '*cliplevel*' standard deviations of the mean pixel value in the box:

- coordinate center of the box
- mean of all valid pixels within the box
- units of the data
- standard deviation of all the valid pixels within the box
- number of valid pixels within the box (used for computing statistics)
- number of pixels within the box deemed not valid based on 'cliplevel'
- minimum of all the valid pixels within the box
- maximum of all the valid pixels within the box

Any number of areas can be selected, one after another, until the skybox command is ended by either clicking the middle button on the mouse, or pressing the 'Return' key on the keyboard. This causes more statistics to be computed:

- the mean of all the means
- the mean of all the standard deviations
- the standard deviation of all the means

For example, the command

```
> sb 9 1
```

draws a $9 \times 9$ pixel box every time the left button is clicked. All the pixels within 1 sigma of the mean pixel value within the box are then used when computing the statistics.

Here's an example of the output format of the **skybox** command:

```
> sb 9 1

Averaging over a 9x9 box.
```

```
|  x        |   y       | mean       |units   |stdev       |pixels|clipped
1h17m25.6s  41d04m29s   2.59294      MJY/SR   0.0442658    54     27
1h17m51.7s  40d08m41s   2.30221      MJY/SR   0.182581     68     13
1h19m22.9s  37d46m38s   2.34462      MJY/SR   0.0644592    53     28
0h57m34.3s  36d17m49s   1.77852      MJY/SR   0.0638208    52     29
0h37m00.5s  35d47m56s   1.79016      MJY/SR   0.0681572    57     24
0h26m56.2s  38d20m00s   1.97641      MJY/SR   0.0412438    56     25
0h27m33.9s  43d25m56s   4.17482      MJY/SR   0.226437     48     33
0h36m02.6s  42d24m15s   2.89665      MJY/SR   0.0883415    54     27
mean of means = 2.48204   mean stdev = 0.0974133  stdev of means = 0.735717
```

### 2.4.11 Cropping Images

The three steps to interactively cropping out a piece of an image are:

Display the image
Define the area to be cropped
Crop the image

**Paint the Image.** Display the image on the screen:

```
> pa image1.fits
```

A window pops up with the given image, **image1.fits** appearing in it. This is the default way of using the paint command. If you aren't familiar with **paint**, see **Section 2.3**.

**Define an Area.** Define the area to be cropped:

```
> da bo
```

will allow me to interactively move a box around on the image to define the area I wish to crop.

```
> da 1 bo
```

will allow me to interactively move a box around on the image to define the area I wish to crop, and assign it the name **Area #1**. This is only one of many ways to define an area with the **define area** command. If you aren't familiar with **define area**, see **Section 2.4.9**.

**Crop the Area.** Crop out the defined area:

```
> cr cropfile
```

crops the last area defined and writes the new image out to a file named **cropfile.fits** (the suffix **.fits** is added if you omit it from the command).

```
> cr 1 cropfile
```

crops the area you previously defined as **Area #1,** in the **define area** example above.

#### 2.4.12 Contour Plots

If you want to make a contour plot of an image, you have 3 choices. All 3 use the same command line syntax, but are invoked by using a different command name, based on the desired output device:

| **command** | **output** |
|---|---|
| contour_plot | Laserprinter |
| contourovl | Overlayed on a displayed image |
| contourps | Postscript file |

Note that the **contour_plot** command can be abbreviated to **contour** or **co**. The **contourovl** and **contourps** commands have no abbreviations.

**Contour Parameters.** The command line parameters are the same for all 3 invocations of the contour plot utility. A description of each parameter follows:

**filename** Name of image to produce a contour plot for. **Default:** Last image displayed.

**start_level** Pixel value to be used for the first (lowest level) contour. **start_level** value can be specified as either a percent (denoted by a % sign) or an absolute data value (e.g., 52, or 1e7). When the percent sign is detected, the program does a histogram and finds the specified percentile point on the histogram. That becomes the first contour level. If no percent sign is detected, the **start_level** is assumed to be an explicit data value, which is then used as the first contour level. **Default:** The pixel value at the 70th percentile level on the histogram of the image.

**constant** The value by which to multiply the data value specified as the **start_level**, for succeeding contours. The formula seeks to produce reasonable levels even when the value of **start_level** turns out to be a negative number. The process continues until 40 contours have been made, or the maximum data value is reached. (value of **constant** gets added if the keyword **linear** is used) **Default:** 1.414

**linear** Scaling of contour levels is arithmetic, instead of geometric. This keyword is used to *add* the value of **constant** instead of multiplying it, thus creating linearly spaced contours. (This is an optional parameter, so **contour** will multiply if this parameter is not used.)

**grid** Optional keyword used to add a coordinate grid to the contour plot.

**gridonly** Keyword used to omit the *contour plot* and only produce the sky coordinate grid. Same effect as using the skyview command, 'grid'.

**Overlaying a Contour Plot on an Image.** Making a contour plot appear in a window on top of an image is easy. First you paint the image.

```
> pa orion.fits
```

To produce the contour plot, use the command:

```
> contourovl
```

A background job will start, leaving you free to continue other work. In a minute or so, the contour plot will appear over the image.

**Hardcopy Plots.** To make a hardcopy contour plot of an image, Type the **contour_plot** command. (Here we will use an abbreviation)

```
> contour image1
```

begins a background job which outputs a contour plot of the file named **image1** to the default printer [12].

[12] See **Section 2.6.5** to set your default printer to an alternate name

**Contour Plots to a Postscript File.** To obtain a contour plot of an image as a Postscript file, use the **contourps** command.

```
> contourps image1
```

**Some Complex Examples.** Here we will show some examples of the use of some other contour parameters. Although we will show the command as **contour_plot**, you could use **contourovl** instead, and get the same results overlaid on the displayed image. The first one

```
> co image1.fits 2.89e7
```

would produce a contour plot of the file named **image1.fits** with the first contour level appearing at the image pixel file value of 2.89e7.

```
> contour image2.fits 60% 200 linear
```

would produce a contour plot of the file named **image2.fits** with the first contour level set at the level below which 60 percent of the pixels lie. The program would then add 200 to that value for the next contour, and on until reaching the maximum data value, or 40 contours.

```
> contour image3.fits grid
```

would produce a contour plot of the file named **image3.fits** with all the default values, and would also overlay a coordinate grid on the plot. The command

```
> contourovl gridonly
```

can be abbreviated

```
> grid
```

When overlaying a grid on a displayed image, it's helpful to put a border of blank pixels around the image, to allow room for coordinates to show.

```
> border
```

By using no parameters with the **border** command, 50 blank pixels are displayed around the perimeter of the image, and the window is resized.[13] You can give any value, or you can specify a different value for each side of the image, if you prefer:

[13]Another use for the **border** command is described in **Section 2.3.8**, under the directions for **zoom** and **pan**.

```
> border 50 100 100 50
```

puts 50 blank pixels along the top of the window, 100 at the bottom, 100 along the left edge, and 50 on the right.

#### 2.4.13 Pixel Editing: The Zap and Write Commands

The **zap** command is used to interactively select out pixels from a displayed image and change their values. When used with no parameters, **zap** enables the mouse for interactive selection of pixels to edit. As the mouse roams the image, the left button should be clicked on each pixel you wish to **zap**.

```
> zap
```

After all the pixels have been selected, click on the middle mouse button, or the <Return> key on the keyboard to end the interactive selection.

The **zap_value**, which is the value you wish the zapped pixels to be given, must be set (using the **set** command), or else the default BLANK value from the FITS header will replace the pixel values. If there is no BLANK keyword in the header, the value for blank pixels can also be **set** (default: -2e+09)[14].

Once you have edited an image, you can create a new image file which includes the edited pixels using the **write** command. The **write** command takes the image file that has had pixels edited on the screen using the **zap** command and writes it out to a new FITS image file on disk. The suffix **.fits** is automatically appended to the filename you specify. So the command:

```
> write pleiades.new
```

would create a new image file on disk called **pleiades.new.fits**, from the last image you edited.

If you have defined an area, using the **da** command, and wish to change the values of all the pixels within the area, the *area* keyword should be specified on the **zap** command line. If you typed the command

```
> zap area
```

All the pixels in the last area you defined would be set to the **zap_value**. Note that currently only the *last* area defined can be zapped. The image can now be written out to a new file, as in the previous example.

[14]For the complete list of rules **skyview** follows when **zap**ping blank pixels, see **Appendix E**.