Lab 3: A Rover's Eye View

Calibration of Mars Pathfinder IMP Images (38 points)

A PDF of the report for this lab must be uploaded to Gradescope by 10 am on Thursday, February 10, 2022. See Lab 1 for submission instructions.

Setup

Back on your lab computer, navigate to your home directory, and create a working directory for the lab called lab03 in your EAPS577 folder. Download all of the files for the lab from this folder into your working directory (remember that unzipping is much faster in the downloads folder before you move the files):
https://www.dropbox.com/sh/kjx1egi1xndj5jw/AADz0-9PtCU3681VR0Rxfvjja?dl=0

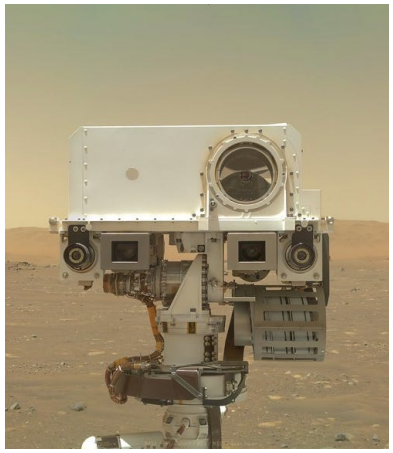
There are two key documents for this lab - this document (lab03-instructions.pdf) gives instructions for the lab, but make sure to also fill out the lab report (lab03-report.docx). You should open and edit this file, and write your answers in all the indicated areas. You will also need to paste your completed code in response to the final question.

Finally, make sure to set up your IDL path every time you start up IDLDE. Go to Window-> Preferences-> IDL -> Paths. Click Insert, and then select your EAPS577 directory (this should include the lab01/lab02/lab03 folders). And make sure to start a journal file now and every time you restart or reset IDL.

Key Concepts/Skills in this Lab:

- Common problems in CCD operation
- The steps involved in calibrating scientific images

1. The raw images



In this lab, you are going to calibrate two images obtained with the Imager for Mars Pathfinder (IMP). The picture on the right shows the IMP on its extendible mast (and below for comparison, its great-great-grandcamera, Mastcam-Z on Perseverance – the rectangles). Click on the link for detailed information on how it works. Note that Pathfinder was the Mars lander, and that Sojourner was the rover it deployed, so we'll be working with images taken by the lander.

What are these images? The raw images here are data arrays with units of digital number (DN) units, which corresponds to the digitized signal read off of the CCD (see lecture for details). In this lab, we are going to apply corrections/calibrations until we

end up with images in radiance units (W m^{-2} μm^{-1} sr^{-1}), which is how much light is reflected from a surface (Watts), per unit area on the surface (m^{-2}), per degree of angle of emission (steradians^{-1}), as a function of wavelength (μm^{-1}). This involves, among other things, bad pixel correction, flat fielding, and absolute responsivity calculation, as we outlined in lecture. The two sets of data are made up of red, green, and blue images, so you will produce two true color images.

The images you will calibrate are in the **`lab3_images.sav`** file. The IDL variables saved in this file are all floating point arrays of size 256 x 248 and are described in **Table 1** below.

Table 1. *The contents of* ***lab3_images.sav***

IDL variables	Contents
rover_red	Raw images of the rover on martian surface using the red/green/blue filters in the right eye of the IMP
rover_grn	
rover_blu	
sky_red	Raw images of the martian sky above a distant peak using the red/green/blue filters in the right eye of the IMP
sky_grn	
sky_blu	

Restore the data and then look at these arrays using help:

```
IDL> help
```

And then please examine all these arrays carefully in ATV2. For example:

```
IDL> atv2, rover_red
```

Note that the images might be upside down – this is because the IMP camera defines the first row at the top, while IDL defines it at the bottom. However, this should be fixed automatically when you write out your images.

2. Test your true color image program

In this lab you'll get to use the programs you wrote in Labs 1 and 2! But first let's make sure that they work with these images (did you appropriately "generalize" your code?).

Using **`write_true_color`** like in Lab 1, save two RGB "true color" images of the rover and sky as **rover_raw.tif** and **sky_raw.tif**.

If the images look a little off, here are a few problems to look for inside of your code:

- Your code should determine the max value rather than using a set number for the stretch
- Your code should find the max value of *all three* images rather than just the red image
- Your code should allow you to specify the file name of the output image

When you are happy with these images, paste them into your lab report. (Question #1a) What potential problems do you observe in these images? List at least two. (Question #1b)

3. Check your bad pixel correction program

Next let's test your pad pixel correction program from Lab 2. Open rover_red in atv2, then from the starting display zoom in once or more. Do you see bad pixels? Open the "Blink" menu and select Blink1. This will store this view in memory. Leave ATV2 open.

Now use **`replace_bad_pixels.pro`** from Lab 2 to make the image look a little better. Create a new array called rover_red_nobad that will contain the corrected image:

```
IDL> rover_red_nobad=replace_bad_pixels(rover_red)
```

Open the new image in ATV2 and check out the results. To directly compare, use blink. Zoom into the same level you did on the original image, set the MouseMode (the button to the left and below the zoom displays) to Blink, then click inside the main display. It should flash to the previous image, and it will stay on the old image if you click and hold. You could also look at the difference between the two images in ATV2 by subtracting them

Evaluate the quality of your corrected image. Did your program fix the obvious bad pixels? Did it overcorrect the image? (Question #2a) Some obvious problems to look for:

- The corrected image looks blurry because you replaced too many pixels with the median, typically indicating that your definition was too harsh.
- The program didn't replace enough pixels because your definition was too lenient.
- The program used set values to define a bad pixel rather than deriving limits from the properties of the image

Make sure your program is telling you how many bad pixels are being corrected – it should include a print command that displays this information, something like:

```
print, 'Bad pixels replaced: ', n_elements(bad_pixels)
```

We'll run this program on all 6 images as part of your calibration pipeline file, so no need to run them manually now.

4. Create your calibration pipeline file

Your goal in this lab is to write one IDL program that will take any IMP color raw image and calibrate it through all of the steps. I've provided a template that you can use for the program, **`imp_cal_template.pro`**, in the shared folder. Make sure to rename this file and add your name/date to the comments at the top.

The first line of the file should start with the following line to define the program and the names of the input variables. In this case, the input variables will be the RGB portions of the image you want to calibrate:

```
PRO imp_cal, red, grn, blu
```

In the program, you don't have to refer to the specific image (e.g., rover_red), you can just refer to the generic variable (e.g., red). The last line in the file should be an END statement: **`END`**

In between these lines we'll build your calibration pipeline. Please make sure to comment at every calibration step. The template includes some basic comments but you should add your own wherever you need to – these notes are both for future you and anyone else who runs the code.

For example, in your program, now copy and paste the command you wrote to output an RGB true color tiff of this new image using write_true_color, and call it **raw.tif**. If you want to get some feedback from your program as it runs, you can add another line after this that prints to the screen:

```
print,'Created RGB image: raw.tif'
```

Try running your program from the command line using these two commands. The first compiles the program, during which IDL will check for any errors and adds the program to the code database, and the second actually runs it. You only have to compile a program once after you make changes. You can also compile by hitting the big compile button in IDLDE.

```
IDL> .com imp_cal
IDL> imp_cal, rover_red, rover_grn, rover_blu
```

Now check you working directory to see if the image appeared. If so, congratulations! Rename it **rover_raw.tif**. Repeat this for the sky image and rename the new image **sky_raw.tif**.

Optional bonus step: If you want to see what your images look like at each step automatically, you can add commands to your pipeline to create a window and use the TV command like in Lab 1 to display the RGB. Just make sure to change the number of the window created at each stage if you want to keep the previous windows up (so window, 0… window, 1 … window, 2 …)

5. Bad Pixel Correction

The next step in your pipeline will be to replace the bad pixels in the raw image. In your pipeline code, create *new images* (something like red_nobad, etc.) with the correction applied, using the command from section 3. Output the result as another true color image, **'nobadpixels.tiff'**.

Now actually run your program on both the rover and sky images, and after each run rename the images so that you have both **rover_nobadpixels.tiff** and **sky_nobadpixels.tiff**. (Question #2b) How many bad pixels did you replace in each of the six raw images? (Question #2c)

6. Dark Current Model

After replacing the bad pixels, the next step in image calibration is to remove various inconsistent behaviors across the CCD. There are many potential sources of pixel to pixel variations in CCD images - intrinsic variations in pixel responsivity, gradients induced by optics, and gradients due to imperfections in filters. These can be wavelength and temperature dependent, and can change with time.

Within your pipeline program, add a line to restore **imp_cal.sav**, which contains all of the files that you'll need to finish off calibration. These are described in Table 2.

Table 2. *The contents of* ***imp_cal.sav***

IDL variables	Contents
dark_lab	***D(x, y)*** **-** Normalized laboratory-measured dark pattern in units of DN/sec
shutter_lab	***S(x, y)*** **-** Normalized laboratory-measured dark shutter pattern in units of DN
ff_red	Normalized flat field for the red/green/blue filters (already corrected for the bad pixels, dark current, and shutter effect)
ff_grn	
ff_blu	

As we discussed in lecture, the raw images that we're starting with are in units of digital number or DN, which is a measure of the number of electrons produced by each CCD element. What are typical DN values for the raw images? (Question #3a, part 1)

Ultimately, our image should only include the electrons that were generated by photon interactions. However, some electrons are generated by other effects in the CCD, and these additional electrons must be **subtracted** from the total DN in your program. This subtraction step is called the "dark current model", and the image that is subtracted is called the **"dark frame"** (the "image" that is generated when no light is actually hitting the CCD). There are three major components to the dark current model, all of which we'll remove with one command:

(1) *Bias* – $\boldsymbol{H_{off}}$ – is the small positive voltage that is always run across the CCD to charge the gates. This is a constant value that must be physically subtracted. For the Pathfinder camera (IMP), **the bias is 8.27 DN**, which must be subtracted from the DN values in the image to remove the bias. How does the bias value compare with the DN values in the image? (Question #3a, part 2)

(2) *Dark current* – ***D(x,y)*** – is due to the electrons generated at a low level even without photon input due to thermal excitations. We can measure this dark current in the lab by taking a "dark image" with the camera shutter closed. Look at the dark_lab image in ATV2. For this filter wheel system, why is there only one dark_lab image, and not one for each R/G/B image? What do you think might cause the dark current to increase toward the top of the image? (*Hint: what increases the amount of dark current? We talked about this in lecture.*) (Question #3b-c)

(3) *Frame transfer smear,* or *electronic shutter effect* – ***S(x,y)*** – is the read noise generated when the signal is read off the CCD. This is mainly due to the charge transfer process across rows and columns, which is *fast* compared to the total readout time, but still takes a finite amount of time (typically 5-10 msec). During this time, incoming photons keep getting detected before the charge is transferred out (unless the camera has a mechanical shutter that is closed prior to this process – most spacecraft imagers do not), and the dark current keeps building up before charge is read out. Thus, this excess charge built up during frame transfer is "smeared" across the array and added to the portion of the image that has not yet been read out. The effect increases linearly away from the read out position, and is extremely scene dependent (Figure 1 left).

While analytic solutions exist for correcting frame transfer smear, the simplest correction approach is to take a subsequent image of the same scene with an exposure time of zero. This zero exposure image only contains signal produced during the frame transfer process by incoming photons and dark current (Figure 1 right). If we subtract that image from the non-zero exposure scene image, we can remove this effect (Figure 1 right inset).

Take a look at the shutter_lab image in ATV2. Do you detect any evidence of frame transfer smear (i.e., "electronic shutter effect") across the raw images, and in which direction (horizontally or vertically)? (Question #3d) Calculate the average percent increase in DN across the image due to this effect. (Question #3e)

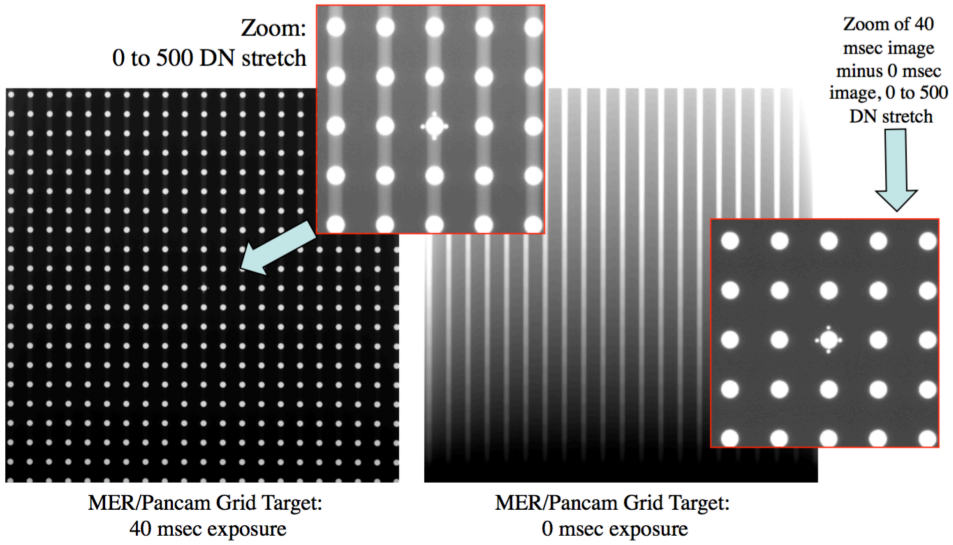
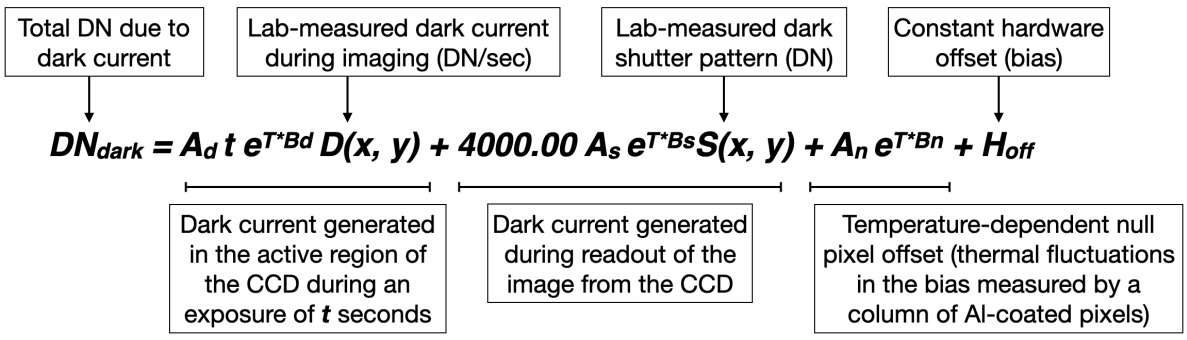


Figure 1: *Frame transfer smear. The left image is a rover image of a simple grid target. If we zoom in and increase the contrast, as shown in the inset, the frame transfer smear along the columns is visible. The right image is the same scene with zero exposure, such that the only signal is being produced by incoming photons and dark current produced during the frame transfer process. The right inset shows the same zoom as before, but after subtraction of the frame transfer smear image (the "shutter" image).*

In a standard camera calibration routine, the dark frame is generated by using a laboratory-derived dark model determined prior to launch [*Reid et al., 1998*]. This model includes all of the effects listed above, with modifications added for the temperature at which the image was taken. Mathematically, the dark current model has the form:



[Note: we have introduced the extra factor of 4000 here for the readout dark current in order to amplify this otherwise subtle effect in the raw data].

So where will you get these numbers?

(1) D(x,y) and S(x,y) correspond to calibration images that you loaded, as shown in Table 2.
(2) Some of these values are constant for all images taken by the IMP instrument:
A_d = 3.016, B_d = 0.105, A_s = 2.845, B_s = 0.105, A_n = 4.05, B_n = 0.144, H_{off} = 8.27
(3) Exposure time and temperature are variables – they depend on the specific observation, as shown in Table 3.

Table 3. *Exposure times and Temperatures for rover and sky images*

Data	Exposure time, ***t*** (*sec*)	Temperature ***T*** (°*C*)
rover_red	0.1080	-17.6433
rover_grn	0.1985	-17.6433
rover_blu	0.1527	-17.6433
sky_red	0.0910	-15.7921
sky_grn	0.1185	-15.7921
sky_blu	0.8105	-15.7921

You will have to add the variables as inputs to your program. Modify the first line of your program as follows (t = exposure time, and temp = temperature):

```
PRO imp_cal, red, grn, blu, t, temp
```

Within your pipeline program, use the formula above to create a dark current model image, and then subtract the dark current model image from each bad pixel corrected red, green, and blue image to create a new image (e.g., **red_nodark**). Then output another true color RGB tiff of the nodark images, named **nodark.tif**.

Use your program to generate dark current corrected images for both the sky and rover, and rename them as **rover_nodark.tif** and **sky_nodark.tif**. (Question #4a)

Look at your corrected images. Did the dark model correct the frame transfer? How can you tell? Describe any residual patterns or artifacts that remain in the images after this step in the processing (experiment with stretching the corrected sky images in ATV2). (Question #4b-c)

7. Flatfield Correction

The dark current model corrects for internal offsets on the CCD, but the external optics and filters can also affect how much light reaches different parts of the CCD, producing their own gradients and other inconsistencies. As we discussed in lecture, we can correct these problems by taking an image of a featureless scene, such as the sky or the inside of an integrating sphere (which simulates isotropic light scattering similar to the sky). This is known as a flatfield image, and any gradients shown in that image can then be **divided out** of subsequent images.

Why is the flatfield image divided out of the scene while the dark current is subtracted? Think

about how changing the brightness of the scene (e.g., a different time of day or a longer exposure time) affects the problems due to dark current vs. those corrected by flatfielding. (Question #5a)

Take a look at the flatfield (ff_red) images, and then divide them out of your dark current corrected images to make a new image called, e.g., **red_noflat**. Then output another true color RGB tiff of the noflat images, named **noflat.tif**.

Use your program to generate flatfield corrected images for both the sky and rover, and rename them as **rover_noflat.tif** and **sky_noflat.tif**. (Question #5b)

Take a look at your flat-field corrected images. Describe some of the improvements (use ATV blink to compare nodark and noflat if needed). Was anything made worse? If so, please elaborate. (Question #5c)

8. Responsivity Correction

Lastly, we have to convert our final "photon" DN values to radiance (reflected light) by taking into account the reponsivity of the CCD. So far, we've only been working in DN, but the absolute responsivity is in units of (DN/sec) / (W m^{-2} μm^{-1} sr^{-1}). Responsivity has the following quadratic form:

$$R = A_1 + A_2T + A_3T^2$$

where T is temperature in °C. Because the CCD response depends on wavelength, the responsivities vary with the filter that was used, as shown in Table 4. Thinking back to lecture, why is the responsivity at blue wavelengths so much less than at red wavelengths? (Question #6a)

Table 4. *Responsivity data for images taken with the three IMP RGB filters.*

Filter	Wavelength (*nm*)	Responsivity parameter, A_1	Responsivity parameter, A_2	Responsivity parameter, A_3
Red	671.2	557.3	-0.575	-0.0014
Green	530.8	578.6	-0.893	-0.0020
Blue	443.2	117.9	-0.392	-0.0006

In your pipeline, define new responsivity variables r_red, r_green, and r_blue that are equal to the calculated responsivity value for each of the red/green/blue images, as a function of temperature (make sure that the constants/parameters you use in calculations are floating point numbers, which you can do be adding a "." after any whole numbers).

The final calibrated data is equal to: **noflat/t/R** (does that make sense to you based on the units?). In your program, add commands to divide each **noflat** image by the responsivity at that wavelength and exposure time, to create your FINAL set of images (e.g., **red_final**). Then output another RGB tiff of the noflat images, named **final.tif**.

Use your program to generate final calibrated images for both the sky and rover, and rename them as **rover_final.tif** and **sky_final.tif**. (Question #6b)

9. The Reveal!

Take a moment to appreciate your excellent work! And next time you see any planetary image, take a moment to appreciate all of the calibration steps and challenging lab measurements that went into making it beautiful.

What color is the Martian sky? What color are the rover wheels (and why)? (Questions #7a-b)

Lastly, copy and paste your finalized working code into your lab report. (Question #8)

Originally written by Jim Bell, expanded and modified by Briony Horgan.