Fiducial Finding

Jeren Suzuki

Last Edited August 27, 2013

Contents

1 Basic Code Overview **1**

2 Outline of Code **1**

3 Detailed Code Overview **1**
3.1 Load Image . . . 1
3.2 Crop image . . . 1
3.3 emboss Filter . . . 1
3.4 shift_diff Filter . . . 2
3.5 Mask Pixels Above Threshold . . . 2
3.6 Calculate Positions of Pixels . . . 2

4 Outline of Math **2**
4.1 emboss . . . 2
4.2 shift_diff . . . 3

5 Using convol() **4**
5.1 Outline of Math . . . 4

6 More Images **5**

7 Exercises **6**

8 Parameters **6**

9 Dealing With Fiducials on Edge **7**
9.1 Images . . . 9

10 Drawbacks **10**

11 How to Break Code **10**

1 Basic Code Overview

In a nutshell, `scrib.pro` takes an image of the sun with fiducials and spits out the positions of the fiducials. How it does this is (in the current iteration of code) it runs two edge-detection filters, isolates pixels above a certain threshold and then calculates what rows/columns the pixels populate most frequently. The result is an array of X and Y positions of the fiducials which we can then orient the sun.

2 Outline of Code

1. Load image
2. Crop image based on known sun center
3. Apply `emboss()` filter
4. Apply `shift_diff()` filter
5. Create mask based on pixels above threshold
6. Calculate rows/columns most populated by pixels
7. Pair the row and column pixels to return fiducial positions

3 Detailed Code Overview

3.1 Load Image

I load a .fits file which has no header; alternatively I could load a .tiff directly. However, if I do this, I have to isolate a single channel since I can't save a tiff explicitly as black and white. Even though the image saved has no color, there are still R, G, and B channels which are all identical. The solution is to slice the 3D array into a 2D Array using any of the R, G, or B channels.

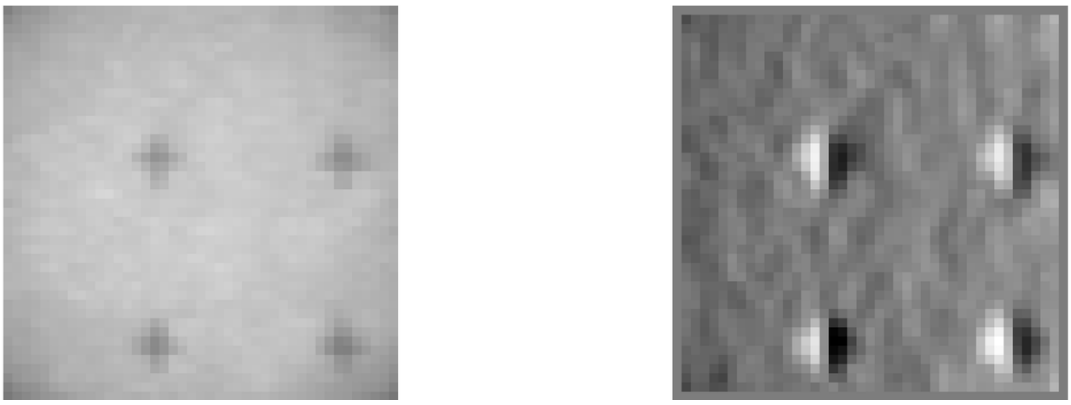
3.2 Crop image

Using values taken from `merrygotrace.pro`, I concentrate on the brightest solar region.

3.3 emboss Filter

The `emboss` filter is a wrapper for IDL's `convol()` function with options to change the convolution kernel. Emboss defaults to look for edges in the x-axis (i.e. left-right direction); you can set an arbitrary "looking" angle. Edges in the original image are straddled with high/low values.

For example, this is an image of an emboss filter in action:



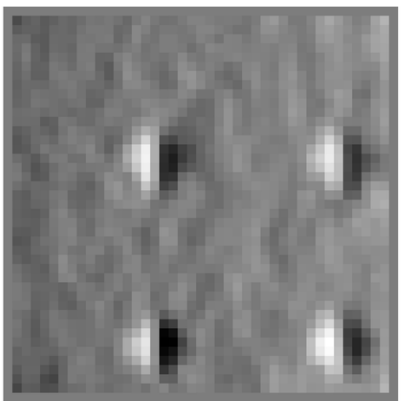
(a) Raw image

(b) Image with `emboss()` filter

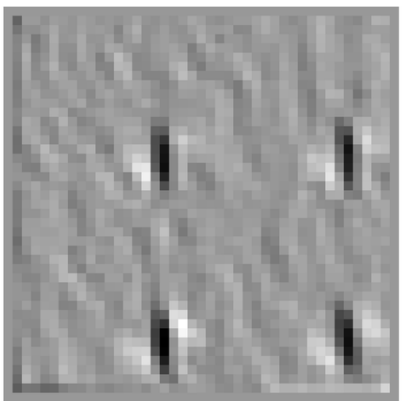
Figure 1: The cropped image we will be looking at

3.4 shift_diff Filter

The `shift_diff` filter takes the `emboss` filtered image and emphasizes the edges created from a large change in a highly negative to a highly positive value (the space between the white and black pixels)



(a) Image with `emboss()` filter

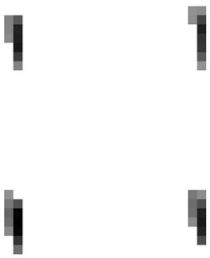


(b) Image with `emboss()` filter and `shift_diff()` filter

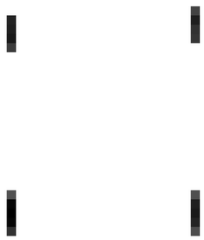
Figure 2

3.5 Mask Pixels Above Threshold

Now we have an image with pixels we know are on the fiducials - we look at pixels above a certain threshold to create vertical and horizontal lines which lie on the fiducials. As of August 27, 2013, the threshold is set to an arbitrary value.



(a) Pixels below -50



(b) Pixels below -80

Figure 3

3.6 Calculate Positions of Pixels

To find the x positions of the pixels I apply a modulo operator to return the remainder of division. (e.g. `8 mod 3 = 2`). To find the y position, I divide the pixel position by the number of rows. I find the mode of each array of X and Y pixel positions to find the most common number (which happens to be on a fiducial). If there is more than 1 fiducial in the image, the mode must be found for one fiducial then without looking at the same row/column twice, find the second most populated row/column.

4 Outline of Math

4.1 emboss

Emboss uses a special kernel:

$$\begin{matrix} 1 & 0 & -1 \\ 1 & 0 & -1 \\ 1 & 0 & -1 \end{matrix}$$

that can be rotated to find edges in a certain direction. For example, `emboss(az=90)` tells the program to look from the 90 degree direction, resulting in a kernel:

$$\begin{matrix} -1 & -1 & -1 \\ 0 & 0 & 0 \\ 1 & 1 & 1 \end{matrix}$$

The angle can be set to any arbitrary value although if the angle is not divisible by 45, the kernel interpolates values between 1 and -1. For `emboss(az=30)`, the kernel looks like:

$$\begin{matrix} 1 & .66 & -.33 \\ 1 & 0 & -1 \\ .33 & -.66 & -1 \end{matrix}$$

4.2 shift_diff

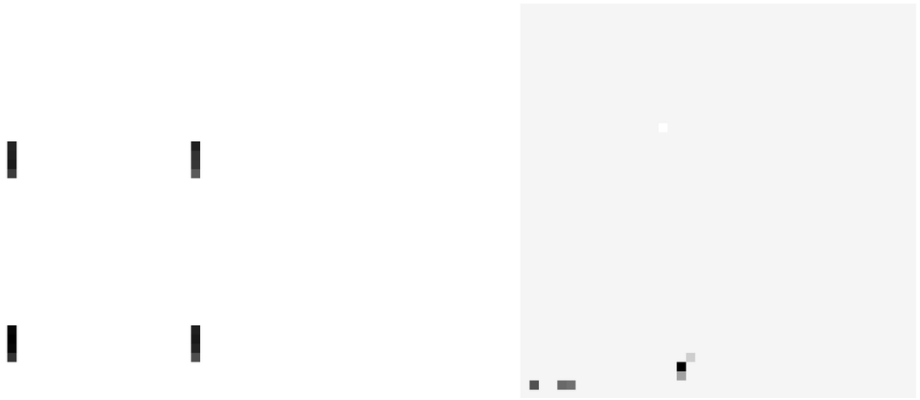
While the kernel can be rotated to any arbitrary angle, we see that rotating the shift_diff kernel is a bad idea.

The shift_diff kernel looks like

$$\begin{matrix} 0 & 0 & 0 \\ 0 & 1 & -1 \\ 0 & 0 & 0 \end{matrix}$$

Where the center value is always 1 and the position of the -1 index can be changed with keywords. Although the position of -1 can be changed, it can have adverse results on the resulting image. The direction of the "looking angle" can be set with a keyword equal to the direction from this matrix:

$$\begin{matrix} 0 & 1 & 2 \\ 3 & x & 4 \\ 5 & 6 & 7 \end{matrix}$$



(a) Pixels below -80 and with `shift_diff(dir=3)`

(b) Pixels below -30 and with `shift_diff(dir=1)`

Figure 4

Basically, if the "looking angle" of `emboss` is perpendicular to the "looking angle" of `shift_diff`, then the resulting image will have no edges. However, we don't always know how rotated the image is beforehand so it is hard to set the right looking angle.

If we expect the fiducials to be rotated only slightly, we can look at the same angle as if the image was not rotated. However, as the angle reaches 90 degrees, we will be increasingly unable to see the fiducials without decreasing the threshold of pixels to mask.

5 Using convol()

Instead of using two edge-detection filters, using a single `convol()` filter with a custom kernel works well enough for isolated fiducials (Note: fails for fiducials cut off on the edge). Once we have a blurred, mostly-symmetric convolved image, we retreieve a mask above a certain threshold and find the center of it with the equation:

$$x = \frac{1}{M}\sum_{i=1}^{n} m_i x_i \ , \ y = \frac{1}{M}\sum_{i=1}^{n} m_i y_i \tag{1}$$

Where M is the total number of pixels we are taking into account and $m_i x_i$ gives us the 'length' of each row/column that we total to find the center. The advantage of this method is that we can get sub-pixel values for the x and y position of the fiducials. The best approach would be to use the current `mode()` method to find the approximate centers of the fiducials then crop a region around it, convolve it, then use the center of mass formula to get a more precise location.

5.1 Outline of Math

The kernel used in the `convol()` filter is:

$$\begin{matrix} 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 \end{matrix}$$

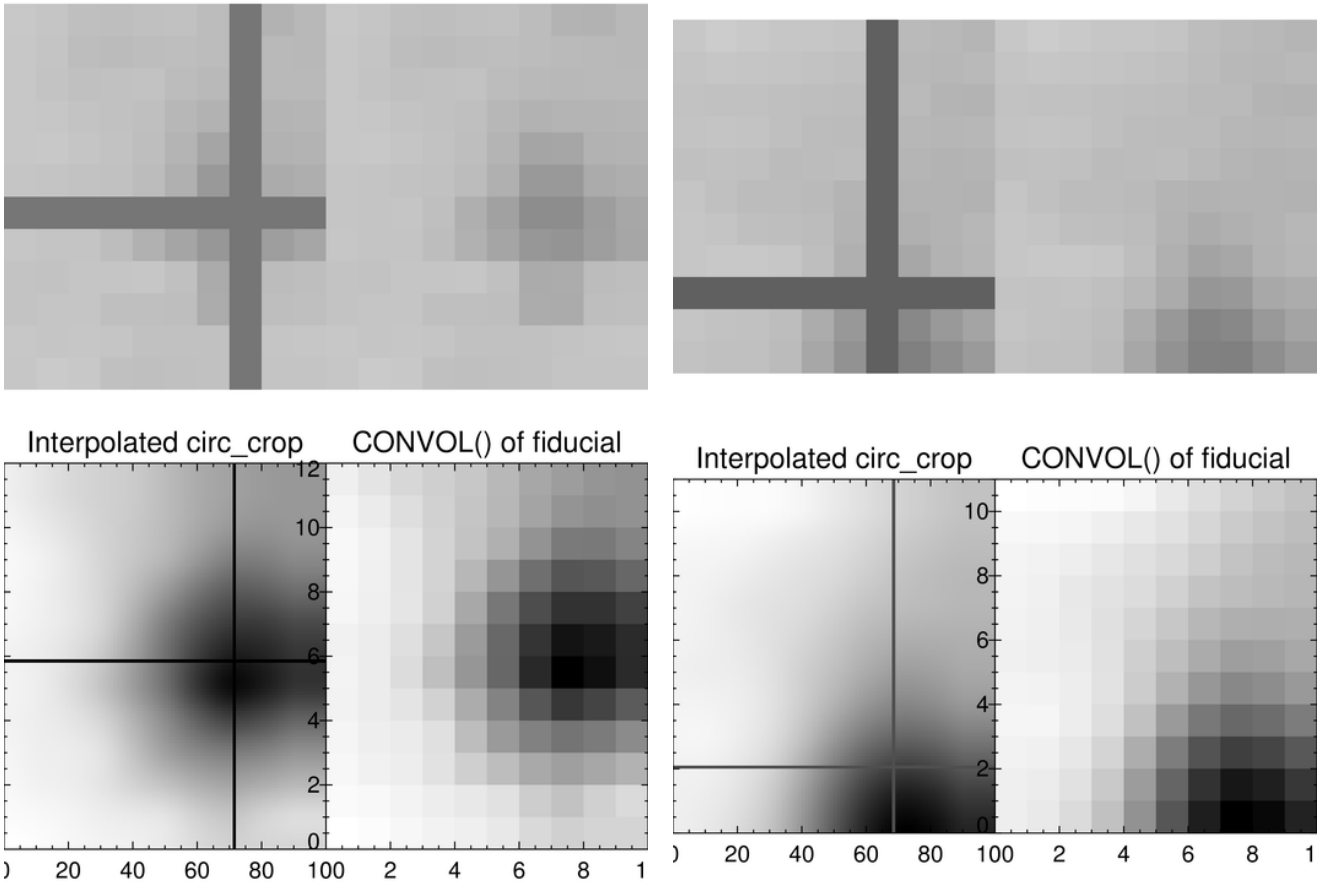
This kernel produces a more symmetric fiducial after convolution than a

$$\begin{matrix} 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \end{matrix}$$

or

$$\begin{matrix} 0 & 1 & 0 \\ 1 & 1 & 1 \\ 0 & 1 & 0 \end{matrix}$$

kernel.



(a) In the top right is the plain image. The bottom right is the plain image convolved with the kernel mentioned in §5.1. The top left is the center of the convolved image using a simple *center-of-mass* formula. The bottom left is the comparison of the center of mass location to an image interpolated to 10x the number of x and y indices.

(b) Here we use the same approach but see that for fiducials cut off by the edge, the simple *center-of-mass* method doesn't work so well.

Figure 5

6 More Images

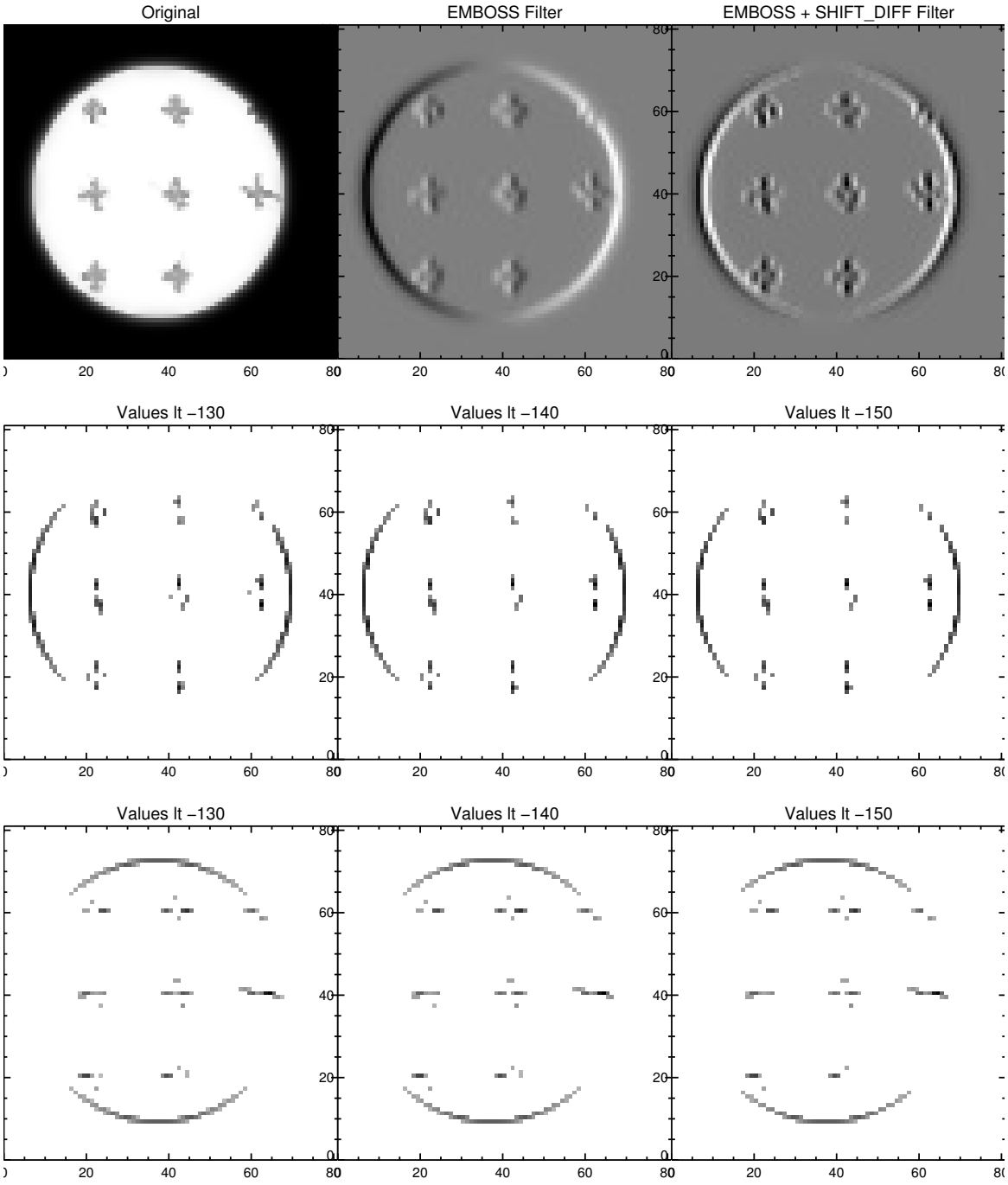


Figure 6: This is a simulated solar image with artificial fiducials. The goal was to see how the filters operated on smudgy fiducials. The results are good. In the second row, the image is filtered with `shift_diff(dir=3)` and in the third row, the image is filtered with `shift_diff(dir=1)`. This was done to improve the number of horizontal and vertical pixels for each direction since the default is at a diagonal.

7 Exercises

Since our attempt to simulate a realistic fiducial proved lacking quantitative reasoning, we looked to extract fiducial values from the real solar image and place them onto a blank (or nearly blank, still uniform) 2d array. We found that even if the fiducial pixel values are inconsistent, as long as they form the shape of the fiducial and are substantially different from the the background value, they will appear to be "perfect". See Figure 9

8 Parameters

In an effort to keep our work rigorous, attempts have been made to minimize arbitrary variables. These attempts have sometimes failed. Here is a list of parameters to quantize:

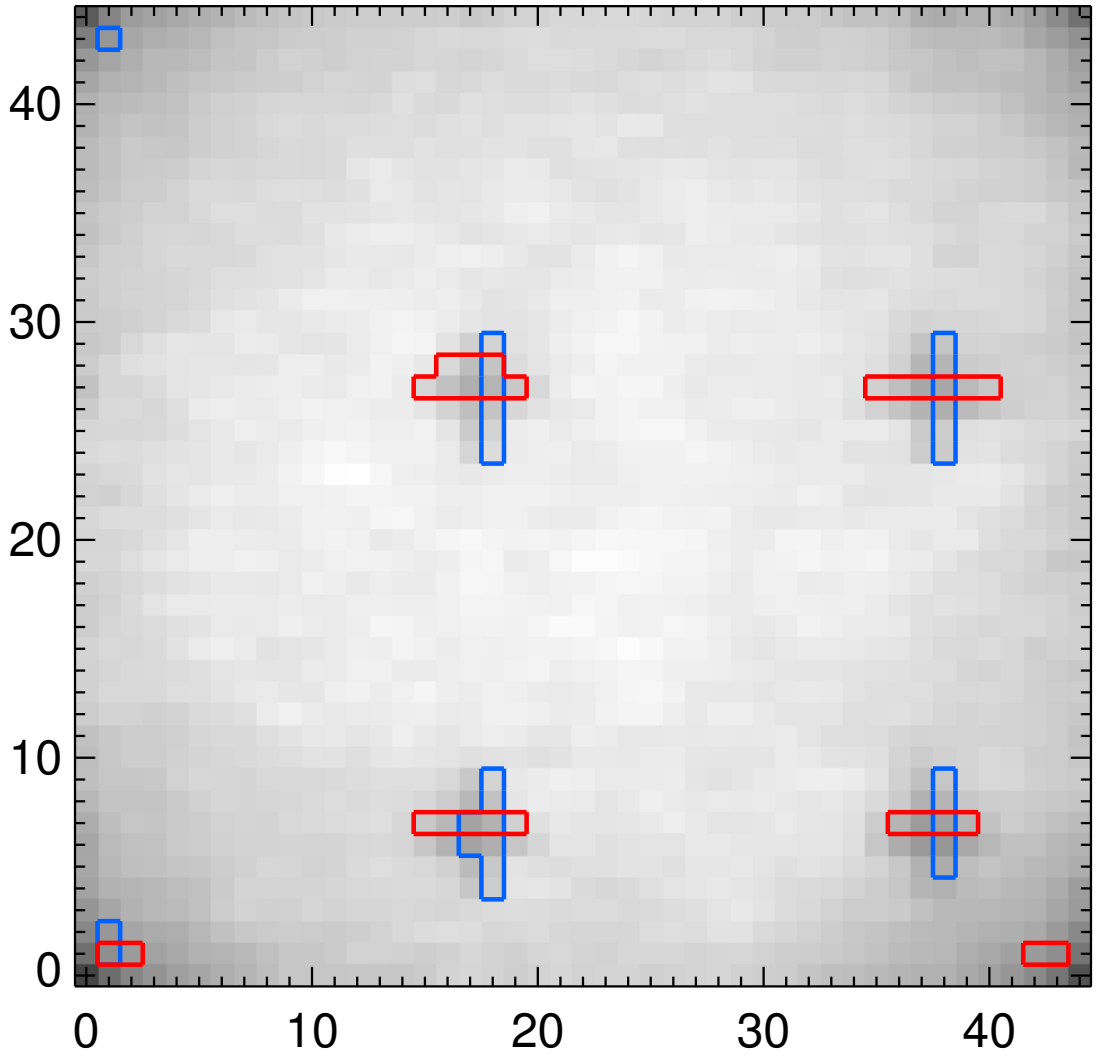


Figure 7: The red outline pixels identify those above a threshold after running the two convolution filters. The rows tend to follow a spatial pattern of identifying the upper row of the 2x2 fiducial center. The blue outlined pixels identify the column pixels which are found using a 90 degree offset in the emboss filter. The position where the red and blue outlined pixels overlap mark the upper right pixel of the fiducial's 2x2 center. Since this is consistent for all 4 fiducials, we can confidently apply it to the dimmer suns. *For now.*

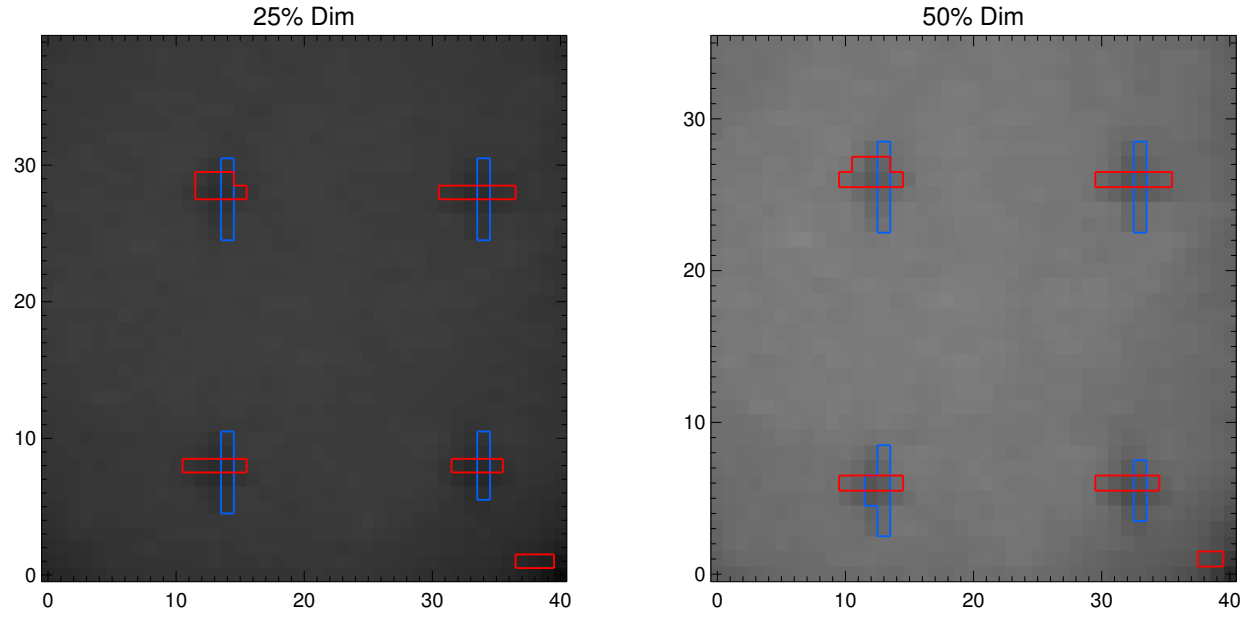
1. Width/Height of cropped box of which to look for fiducials in. Must be large enough to cover as much area of the sun as possible without including the solar limb.
2. Threshold used to mask twice-convoluted image. As of August 27, 2013, arbitrary.
3. ...and others once we recognize them.

9 Dealing With Fiducials on Edge

Sometimes when cropping the inner part of the sun, the cropped region will inadvertently cut off a fiducial. There are two ways of dealing with this:

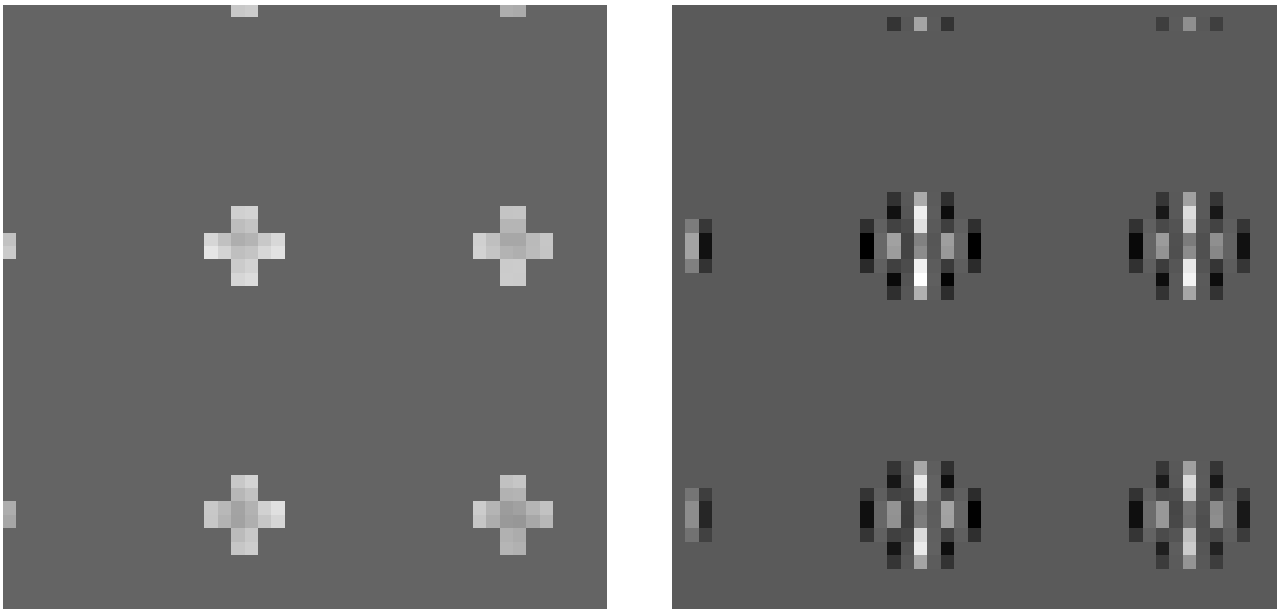
1. Crop so that fiducials aren't cut off
2. Deal with cut off fiducials

We are focusing on the latter. However, this in turn brings forth another problem; do we take an image with cut-off fiducials (CFs) and then eliminate them or do we try to find fiducials that aren't cut off, i.e., look around



(a) (b)

Figure 8: By varying the threshold on the dimmer suns, we get the same masking pattern. For the 50% dimmer sun, we use half the threshold value used for the 100% sun. For the sun at 25% the brightness of the main sun, we use a quarter of the threshold.

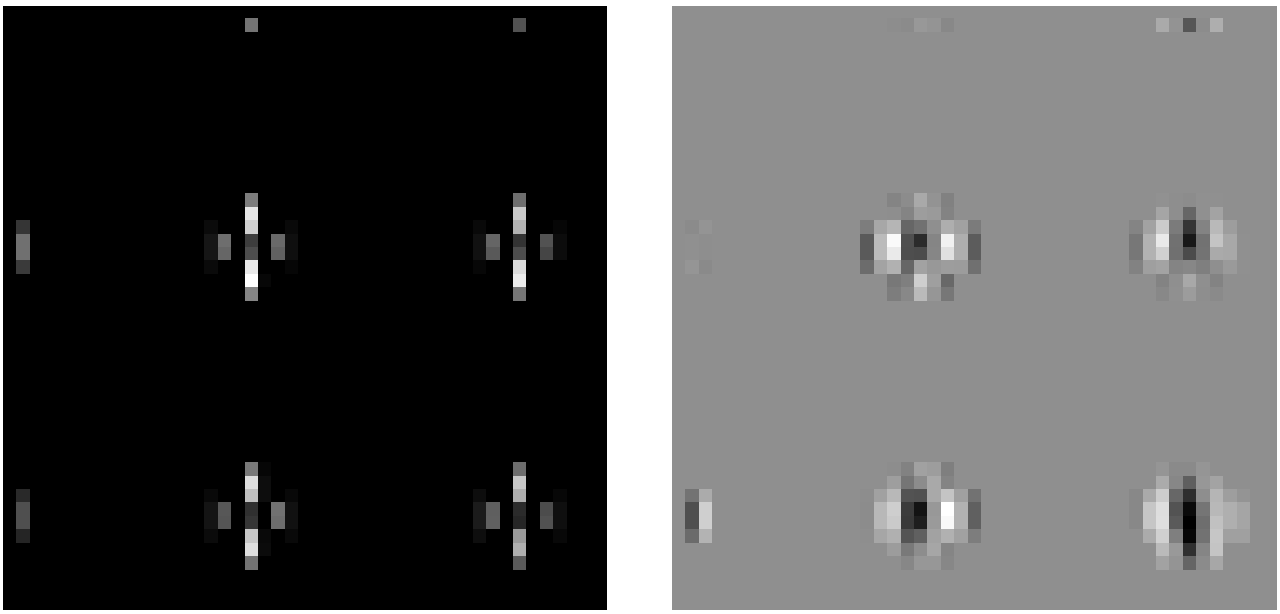


(a) (b)

Figure 9: On the left we see the real fiducial values over a uniform 2D array. On the right is the result of the two convolution filters. Despite the "blurry" pixel values, the filters saw the edges of the fiducial.

CFs? Our current method is to find the fiducials normally, as if the fiducials aren't cut off, then methodically eliminate any fiducials that are too close to the edge of our image. The result is that we need to identify the shape and size of a fiducial beforehand to apply those constraints on the on-edge or CFs. Also, this means that we need to identify "how bad" we will allow the fiducial finding program to continue finding CFs.

Also, if we see a CF on the edge, instead of cropping the image so that the CF is no longer in the image, we simply tell the program to ignore anything it finds on that side of the image. To do this, we look at a 1 pixel border around the edge of our image and detect if there are any fiducials there to begin with. If there are, we look towards the inside 6 pixels deep. The reasoning behind 6 pixels is that if we look 6 pixels to the center and "see" a complete fiducial, then it isn't cut off and only the edge of the fiducial lies on the edge of the image. If we look 6 pixels in and find any number of fiducial pixels less than 6, then we know that the fiducial is cut off.



(a) (b)

Figure 10: On the left is the same image as Figure 12(b), but masked to values above a certain threshold. Here you see that there clearly exists a linear structure. I also tried instead of a gray background, a background that was equal to the most common pixel value (198 instead of 100). The goal was to try and blur out the fiducial as much as possible. On the right we see the convolution filter has a harder time finding the edges but we still see some clear vertical black lines that we can mask out.

Now, the problem is how to look for a fiducial. Currently, the imperfect method is to look at the pixel values themselves and try to find some pattern within them to isolate a fiducial.

9.1 Images

In Figure **??**, the vertical line represents the left edge of a fiducial that is cut off on the right side of our image. Anything that is to the left of the vertical line we do not directly care about since it is not a fiducial, but we still need it for derivatives and such. Anything to the right of the vertical line represents a portion of the fiducial that is cut off in our image. The reasoning is that we look at a set of decreasingly cut off fiducials and try to isolate a pattern. The order of decreasing cut off goes from left to right, top to bottom. In the top left we see 1 pixel of the fiducial in the image and in the bottom right we see 6 pixels of the fiducial in the image; the fiducial is not cut off.

After a cursory analysis at the plots we see that there is no obvious pattern for CFs. This can be remedied by either of 2 methods:

1. Look at more pixels
2. Look in a different manner
3. A combination of the above

After a second, third, and Nth look, there exists a threshold to isolate fiducials. See Figure **??**.

If we use a threshold to count the number of pixels below (in this case) -20, we can count how many pixels are in a fiducial. For example, the top left image in Figure **??** only has 1 pixel below the threshold which corresponds to the number of pixels of the fiducial we see. In the 2nd from top image in the right-most column, we see that there are 4 pixels below the threshold of -20 which agrees with 4 pixels to the right of the vertical line. Using this method, we can accurately threshold pixels so that we can eliminate any that are cut off. The downside however is that the threshold varies from fiducial to fiducial. For example, in Figure **??**, we see that the threshold is at -30.

After applying the same techniques above to 2 more fiducials, the thresholds are approximately -30 as well. I also thought that perhaps whether we were looking in rows/columns affected the threshold somehow, but I looked at the same ficudial both directions and found the threshold to maintain around -30.

10 Drawbacks

The main problems facing this program are:

1. With rotated fiducials above 2 degrees, cannot use `mode()` to easily find most common pixel positions
2. With rotated fiducials (and without knowing what angle they are rotated at beforehand), we may have to try multiple rotation angles for the kernels used in each filter.
3. If the fiducials are not plus-shaped the edge-detection tools cannot handle distortion. Ideally, a circle-shaped fiducial would be fine except that any distortion returns poor results with the edge detection filters. I think this is because the plus-shaped fiducials naturally have up and down edges which, even with distortion, can still be isolated. See Figure 11
4. We must crop the sun in such a way that none of the solar limb is in the image. The reason is because the image is pixelated enough that the edge of the sun is detected as an edge, albeit one we do not want. A problem that arises from this is that when we crop out the solar limbs, we may cut off fiducials.
5. If the fiducial is too close to the edge of the cropped image, then problems arise. There are two ways to solve this. We can tell the program to only look for complete fiducials or 2) "zero" out the incomplete fiducials so the program finds fiducials as normal.
6. In `circscancrop`, the thresholds are arbitrary. To scan for region 2, (i.e., the 50% brightness sun) we have the threshold set to 0.3 of the sorted `max(image)`. For region 3 (i.e., the 25% brightness sun) the threshold is set to 0.2 of the sorted `max(image)`. I tried making the threshold 0.3 of the image with the highest 1% pixels eliminated, but the change in threshold made the circle-scanning break. Either I need to find a new way to parameterize the thresh or change the circle-scanning code.
7. The fiducial positions for the 25% sun seem to be wrong. This is either because of a wrong center position or the fiducial finding code not working properly (inclined to agree with first result, 100% and 50% brightness suns look identical, must be something wrong with the 25% brightness sun. Right?)

11 How to Break Code

1. Sun is too close to the edge of the solar image
   - When cropping, either need to fabricate null data or ignore altogether
   - Now looks at the image border and if it counts a certain number of pixels above a threshold, the image should be ignored.
   - Fixed it (April 9, 2013)
2. Part of sun is cut off in image
   - Not only will cropping fail, limb fitting will also fail
   - Same as above
   - Fixed it (April 9, 2013)
3. Bad pixels
   - Threshold finding has been changed to remedy this
4. When scanning in a circle, haven't completely coded fail-safes if it doesn't find auxiliary suns
   - Probably have to do something where if I don't detect the auxiliary suns, I'll scan at different radii at specific intervals
5. Image is rotated more than 2 degrees
   - Current fiducial finding method can't handle a rotated image well
   - GRIPS proposal says the pendulum will swing up to ± .1 degree; is this the same rotation parameter in our image?
   - Don't have to worry about this anymore
6. Fiducial is too close to edge
   - If a fiducial is too close to edge of cropped image, incomplete fiducial shape confuses program
   - Not sure how to fix yet
   - Trying a border mask
   - Problem addressed in Section 9

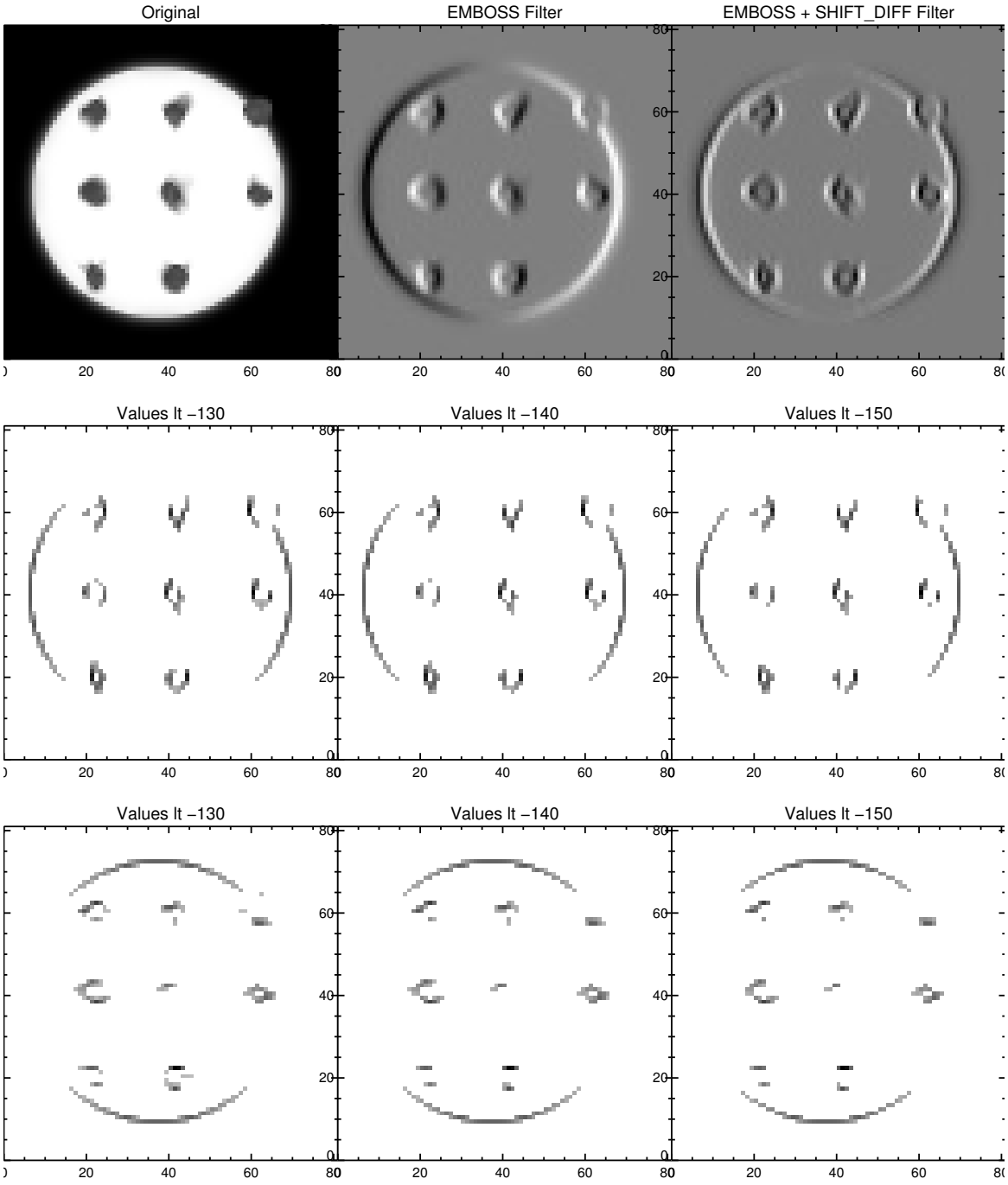


Figure 11: The fiducials are initially circles but with a bit of distortion we see the typical method to find fiducials no longer works. Perhaps with circle-shaped fiducials we could apply some form of Gaussian fitting but using plus-signs instead works better since it's hard to calculate the rotation angle from a bunch of distorted circles.

7. Scanning in a circle at high resolution (higher than 1 pixel a degree) makes the program break, although I'm not sure why
   - Fixed it (Mar 13, 2013)