Fiducial Finding

Jeren Suzuki

Last Edited February 20, 2013

Contents

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:

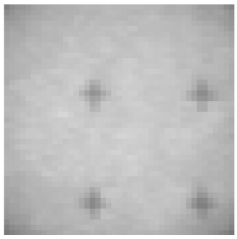

Figure 1: Raw image

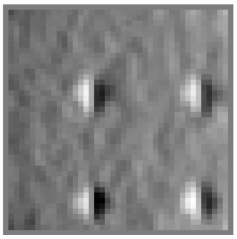

Figure 2: Image with `emboss()` filter

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 toa highly positive value (the space between the white and black pixels)

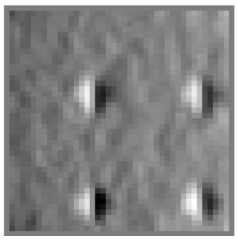

Figure 3: Image with `emboss()` filter

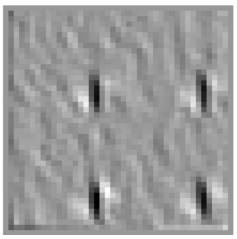

Figure 4: Image with `emboss()` filter and `shift_diff()` filter

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.

Figure 5: Pixels below -50

Figure 6: Pixels below -80

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 numer 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 multiple times.

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 abitrary 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:

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

Figure 7: Pixels below -80 and with `shift_diff(dir=3)`

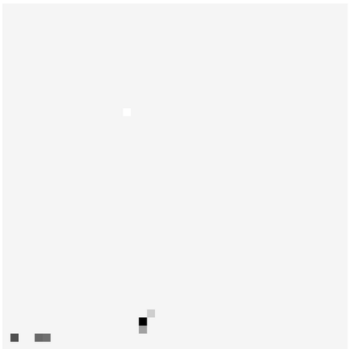

Figure 8: Pixels below -30 and with `shift_diff(dir=1)`

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 More Images

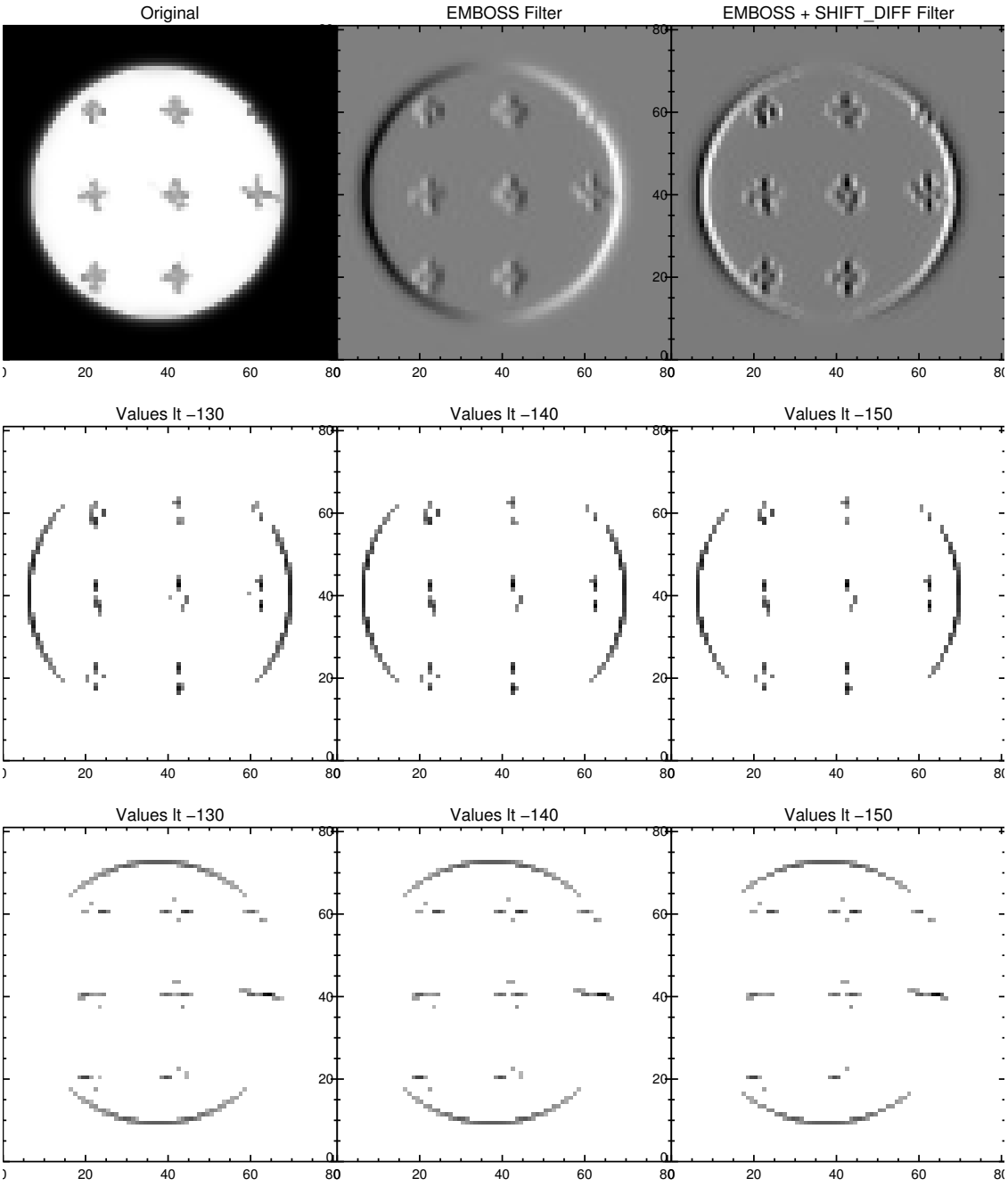

Figure 9: This is a simulated solar image with artificial fiducials. The goal was to see how the filters operated on smudgy fiducials. The results are good. In the second row, the image is filtered with `shift_diff(dir=3)` and in the third row, the image is filtered with `shift_diff(dir=1)`. This was done to improve the number of horizontal and vertical pixels for each direction since the default is at a diagonal.

6 Drawbacks

The main problems facing this program are:

1. With rotated fiducials, 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 fidcial 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 10

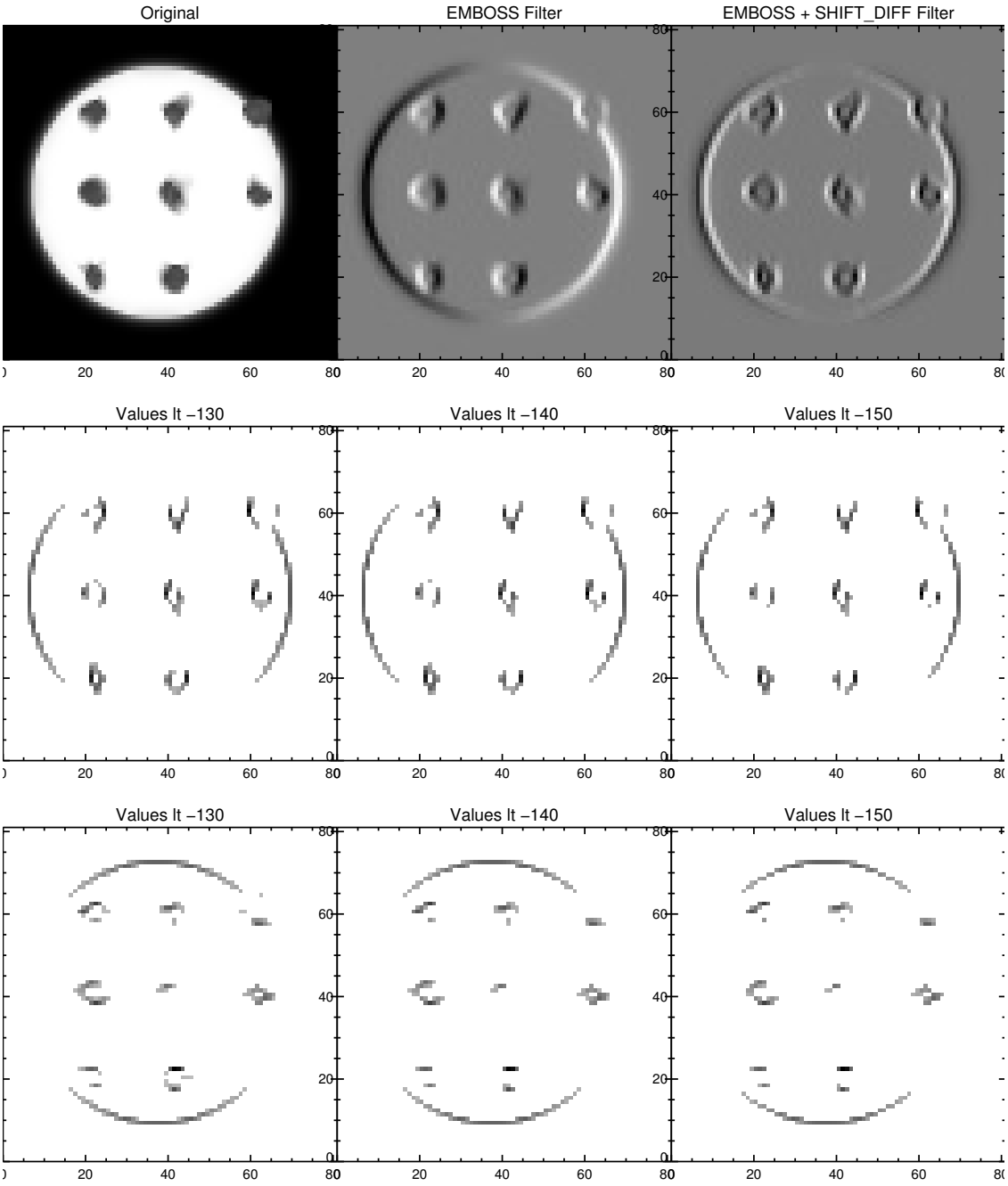

Figure 10: The fiducials are initially circles but with a bit of distortion we see the typical method to find fiducials no longer works. Perhaps with circle-shaped fiducials we could apply some form of gaussian fitting but using plus-signs instead works better since it's hard to calculate the rotation angle from a bunch of distorted circles.